Composition dynamique de services sensibles au contexte dans les systèmes intelligents ambiants

Ali Yachir

THESE EN COTUTELLE

Présentée pour l'obtention du :

Grade de Docteur de l'Université Paris-Est
(Spécialité : Informatique)

Grade de Docteur En Sciences de l'USTHB
(Spécialité : Informatique)

Par : YACHIR Ali

COMPOSITION DYNAMIQUE DE SERVICES SENSIBLES AU CONTEXTE DANS LES SYSTEMES INTELLIGENTS AMBIANTS

Soutenue publiquement, le 23 Février 2014, devant le jury composé de :

Rapporteurs : **M. Olivier BOISSIER**, Professeur, Ecole Nationale Supérieure des Mines de Saint-Etienne, France.

M. Djamel BENSLIMANE, Professeur, Université Lyon 1, France.

Directeurs : **M. Yacine AMIRAT**, Professeur, Université Paris-Est Créteil (UPEC, ex. Paris12), France.

M. Nadjib BADACHE, Professeur, Université des Sciences et de la Technologie Houari Boumediene (USTHB), Algérie.

Examinateurs : Mme. Aicha AISSANI MOKHTARI, Professeur, Université des Sciences et de la Technologie Houari Boumediene (USTHB), Algérie.

Mme. Samira MOUSSAOUI, Professeur, Université des Sciences et de la Technologie Houari Boumediene (USTHB), Algérie.

M. Abdelghani CHIBANI, Maître de Conférences, Université Paris-Est Créteil (UPEC, ex. Paris12), France.

M. Jean-Yves TIGLI, Maître de Conférences, Université de Nice-Sophia Antipolis, France.

Résumé

Résumé : Grâce à l'apparition des paradigmes de l'intelligence ambiante et de la robotique ubiquitaire, on assiste à l'émergence de nouveaux systèmes intelligents ambiants visant à créer et gérer des environnements ou écosystèmes intelligents d'une façon intuitive et transparente. Ces environnements sont des espaces intelligents caractérisés notamment par l'ouverture, l'hétérogénéité, l'incertitude et la dynamique des entités qui les constituent. Ces caractéristiques soulèvent ainsi des défis scientifiques considérables pour la conception et la mise en place d'un système intelligent adéquat. Ces défis sont principalement au nombre de cinq : l'abstraction de la représentation des entités hétérogènes, la gestion des incertitudes, la réactivité aux événements, la sensibilité au contexte et l'auto-adaptation face aux changements imprévisibles qui peuvent se produire dans un environnement ambiant. L'approche par composition dynamique de services constitue l'une des réponses prometteuses à ces défis. Dans cette thèse, nous avons proposé un système intelligent capable d'effectuer une composition dynamique de services en tenant compte, d'une part, du contexte d'utilisation et des diverses fonctionnalités offertes par les services disponibles dans un environnement ambiant et d'autre part, des besoins variables exprimés par les utilisateurs. Ce système est construit suivant un modèle multicouche, adaptatif et réactif aux événements. Il repose aussi sur l'emploi d'un modèle de connaissances assez expressif lui permettant une ouverture plus large vers les différentes entités de l'environnement ambiant notamment : les dispositifs, les services, les événements, le contexte et les utilisateurs. Ce système intègre également un modèle de découverte et de classification de services afin de localiser et de préparer sémantiquement les services nécessaires à la composition de services. Cette composition est réalisée d'une façon automatique et dynamique en deux phases principales: la phase offline et la phase online. Dans la phase offline, un graphe global reliant tous les services abstraits disponibles est généré automatiquement en se basant sur des règles de décision sur les entrées et sorties des services. Dans la phase online, des sous graphes sont extraits spontanément à partir du graphe global selon les tâches à réaliser qui sont déclenchées par des événements qui surviennent dans l'environnement ambiant. Les sous graphes ainsi obtenus sont exécutés suivant un modèle de sélection et de monitoring de services afin de tenir compte du contexte d'utilisation et garantir une meilleure qualité de service. Les différents modèles proposés ont été mis en œuvre et validés sur la plateforme ubiquitaire d'expérimentation du laboratoire LISSI à partir de plusieurs scénarii d'assistance et de maintien de personnes à domicile.

Mots clés : Informatique ubiquitaire, informatique ambiante et diffuse, robotique ubiquitaire, intelligence ambiante, systèmes intelligents ambiants, composition de services, sélection de services, monitoring de services, qualité de service, sensibilité au contexte, auto-adaptation.

Abstract

Abstract: With the appearance of the paradigms of ambient intelligence and ubiquitous robotics, new ambient intelligent systems are emerging with the aim to create and manage intelligent environments or ecosystems in an intuitive and transparent way. These environments are smart spaces characterized mainly by the openness, the heterogeneity, the uncertainty and the dynamic of the entities that constitute them. These characteristics involve significant scientific challenges for the design and implementation of an appropriate intelligent system. These challenges are mostly heterogeneity abstraction, uncertainty management, reactivity to events detection, context-awareness and self-adaptation to unpredictable changes that may occur in an ambient environment. The approach by dynamic service composition is one of the promising keys to address these challenges. In this thesis, we proposed an ambient intelligent system capable of performing dynamic services composition by taking into account, on the one hand, the context of use and the various functionalities offered by the available services in the ambient environment and, on the other hand, the varying needs expressed by users. This system is built in a multilayer model which is adaptive and reactive to events detection. It is also based on a knowledge model enough expressive offring a larger openness to the various entities of the surrounding environment including: services, events, context and users. This system also includes a model of services discovery and classification to locate and semantically prepare the necessary services for the process of services composition. This composition is performed in an automatic and dynamic manner in two main phases: the online phase and the offline phase. In the offline phase, a global graph connecting all the available abstract services is automatically generated based on decision rules on the inputs and outputs of services. In the online phase, subgraphs are extracted spontaneously from the global graph according to the tasks to perform that are triggered by events detected in the surrounding environment. The resulting subgraphs are executed in accordance with a services selection and monitoring model to take into account the context of use and guarantee a better quality of service. The various proposed models have been implemented and validated on the ubiquitous experimental platform of LISSI laboratory through several scenarios for assistance and keeping people at home.

Keywords: Ubiquitous computing, ambient and pervasive computing, ubiquitous robotics, ambient intelligence, ambient intelligent systems, services composition, services selection, services monitoring, quality of service, context-awareness, self-adaptation.

Dédicaces

À mes parents ;

À ma femme;

À mon petit fils Wassim ;

À toute ma famille ;

À tous ceux qui me sont chers.

Remerciements

Louange à DIEU qui m'a donné force, courage et patience afin d'accomplir ce travail de thèse dans de très bonnes conditions.

Je remercie tout d'abord mes Directeurs de thèse, M. le Professeur Yacine AMIRAT et M. le Professeur Nadjib BADACHE pour leur soutien tout au long de ce travail. Je suis très heureux de pouvoir leur exprimer mes plus vifs remerciements, ma profonde gratitude et ma très sincère reconnaissance.

Je remercie M. Olivier BOISSIER, Professeur à l'Ecole Nationale Supérieure des Mines de Saint-Etienne (France), et M. Djamel BENSLIMANE, Professeur à l'Université Lyon 1 (France), de m'avoir fait l'honneur d'accepter d'être rapporteurs de cette thèse.

Je tiens à remercier également Mme. Aicha AISSANI MOKHTARI, Professeur à l'Université des Sciences et de la Technologie Houari Boumediene (USTHB), Mme. Samira MOUSSAOUI, Professeur à l'USTHB (Algérie), M. Abdelghani CHIBANI, Maître de conférences à l'Université Paris-Est Créteil (UPEC, ex. Paris12), et M. Jean-Yves TIGLI, Maître de conférences à l'Université de Nice-Sophia Antipolis (France), qui m'ont fait l'honneur d'avoir accepté de faire partie du jury.

Je remercie très vivement ma femme qui m'a beaucoup aidé et rendu le déroulement de cette thèse agréable ainsi que tous les membres de ma famille pour la confiance qu'ils m'accordent, leur patience, leur amour, et leurs encouragements incessants.

Mes remerciements vont naturellement à tous mes amis et collègues, particulièrement ceux de l'Ecole EMP d'Alger et de l'université USTHB. Je leur exprime ma profonde sympathie et je leur souhaite une bonne continuation.

Enfin, un grand merci à tous les amis et collègues que j'ai côtoyés durant mes séjours au Laboratoire Images, Signaux et Systèmes Intelligents (LISSI) de l'Université Paris-Est Créteil (UPEC, ex. Paris12), et qui m'ont bien accueilli.

Table des matières

Table des figures

Liste des tableaux

CHAPITRE 1

Introduction et motivations

1.1. Introduction générale

Grâce aux technologies d'identification par radio-étiquettes (tags) RFID et aux réseaux de communication sans fil à faible consommation électrique, des objets communicants représentant un capteur, un actionneur ou un objet physique quelconque (TV, Smartphone, tablette tactile, etc.), deviennent accessibles à large échelle selon le paradigme de l'Internet des objets. Cette déclinaison thématique de l'informatique ubiquitaire est appelée, intelligence ambiante ou Ambient Intelligence (AmI). A travers ce paradigme, il s'agit d'exploiter ces objets communicants pour fournir à l'utilisateur, en tout lieu et à tout moment, des services d'assistance, de communication et d'information. De même, utiliser des robots dans des environnements à intelligence ambiante, comme fournisseurs de services ou consommateurs de services fournis par d'autres objets, définit le concept de la robotique ubiquitaire. Avec l'apparition des paradigmes de l'intelligence ambiante et de la robotique ubiquitaire, on assiste à l'émergence de nouveaux systèmes intelligents ambiants visant à créer et gérer des environnements ou écosystèmes intelligents d'une façon intuitive et transparente. Ces systèmes, appelés systèmes AmI, constituent l'évolution naturelle des systèmes de traitement et d'accès à l'information centrés sur l'utilisateur humain. Cet accès à l'information est totalement intégré dans l'environnement physique et se veut transparent dans sa manipulation. L'objectif visé par les systèmes AmI consiste à offrir une multitude de fonctions et de services accessibles, à n'importe quel moment, et à n'importe quel endroit, et selon une multitude de modes d'interactions et de média. La particularité de ces systèmes réside dans leur capacité à adapter continuellement et automatiquement la même fonction ou service, aux différents contextes et besoins exprimés explicitement ou implicitement par les utilisateurs. Ainsi, la dimension d'intelligence d'un système AmI réside dans sa capacité d'une part, à percevoir et analyser l'environnement et les besoins des utilisateurs et d'autre part, à fournir des services répondant à ces besoins et à même proposer de nouveaux services à valeur ajoutée pour les utilisateurs. Ainsi, dans le cadre des applications de maintien à domicile des personnes âgées ou dépendantes, un système à intelligence ambiante peut offrir une multitude de services réactifs ou proactifs permettant d'améliorer la qualité de vie et l'état physique, mental, et le bien-être social des usagers. Ces services peuvent être de plusieurs types : Assistance à la mobilité, assistance cognitive, sécurité, surveillance médicale, maintien du lien social, etc.

Cependant, la mise en œuvre des concepts de l'intelligence ambiante et de la robotique ubiquitaire par la construction de systèmes AmI adéquats soulève des défis scientifiques considérables. En effet, les environnements quotidiens des utilisateurs sont composés d'équipements variés et de natures distribuée et hétérogène. En plus de l'hétérogénéité des dispositifs, ces environnements sont très ouverts, hautement dynamiques et incertains.

La nature dynamique est due aux changements fréquents qui se produisent dans les environnements ambiants alors que la nature incertaine est due à l'apparition de ces changements d'une façon aléatoire et imprévisible. Ces changements continuels sont liés à la volatilité des objets présents (ajouts et retraits d'objet, mobilité dans l'espace) et à la mobilité des utilisateurs et à la variation de leurs besoins. Par ailleurs, les divers équipements qui se trouvent dans un environnement ambiant peuvent se connecter et se déconnecter d'une façon imprévisible au gré de leurs déplacements ou en raison de pannes ou de mesures de sauvegarde d'énergie. Par conséquent, de nouvelles fonctionnalités sont offertes par les dispositifs qui apparaissent et d'autres fonctionnalités sont supprimées à cause des dispositifs qui disparaissent. Ce dernier cas peut causer une dégradation conséquente de la qualité fourni par un système AmI. Cette dégradation peut aller d'une simple indisponibilité momentanée de certaines fonctionnalités jusqu'à leurs disparitions définitives en cas de pannes majeurs.

Certaines réponses à ces défis sont aujourd'hui offertes par de nouvelles approches intergicielles qui permettent de masquer l'hétérogénéité et la distribution des différentes entités des environnements ambiants. De plus, la forte dynamicité de ces environnements impose une flexibilité des applications et un aspect lâche du couplage entre les entités impliquées. L'orientation service est un paradigme qui constitue l'une des réponses prometteuses à cette demande. Dans cette thèse, nous considérons que l'approche orientée service demeure la base de la construction d'un système AmI en raison de l'abstraction apportée dans la représentation des entités de l'environnement ambiant et du couplage lâche (faible) qu'elle permet entre ces différentes représentations. En effet, les entités hétérogènes ne sont représentées que par leurs fonctionnalités encapsulées dans la notion de service qui est indépendante des technologies d'implémentation. Ainsi, le paradigme orienté service garantit l'interopérabilité et le couplage faible nécessaires à la construction dynamique d'applications dans un environnement ouvert et hétérogène. En représentant les différents dispositifs de l'environnement ambiant comme des fournisseurs et des consommateurs de services, le problème de la construction dynamique d'applications peut être ainsi traité comme un problème de composition dynamique de services.

La composition dynamique de services est la construction automatisée d'un service composite à partir de services composants. Les services composites résultants fournissent de nouvelles fonctionnalités qui ne sont pas directement disponibles dans les services composants élémentaires. Ainsi, la composition dynamique de services correspond à la recherche d'une combinaison des services élémentaires les mieux adaptés en réponse à l'expression de besoins par les utilisateurs. La combinaison de services disponibles représente un service composite complexe réalisant les fonctionnalités requises. Le choix des composantes et l'établissement des liens de communications entre ces dernières pour la réalisation d'une application de plus haut niveau s'effectue sans intervention humaine. En effet, devant la prolifération du nombre de services disponibles, il est au-delà de la compétence humaine d'analyser et de générer des services composites manuellement. Le processus est totalement pris en compte par le système qui peut décider en fonction d'évènements particuliers de créer ou de modifier les composites. Á ceci, s'ajoutent les informations sur le contexte et la qualité des services qui sont des éléments importants à prendre en compte lors du processus de composition de services afin d'améliorer les performances du système. Par ailleurs, l'ensemble du processus de

composition dynamique doit supporter la volatilité des objets et la mobilité des utilisateurs tout en assurant une continuité de services par une adaptation à la volée, i.e. sans arrêter son fonctionnement.

Ainsi, la composition dynamique de services constitue le noyau d'un système AmI réalisé par une approche orientée services. Il s'agit du mécanisme responsable de la construction dynamique d'applications en tenant compte d'une part, des fonctionnalités offertes par l'ensemble des services disponibles et du contexte d'utilisation et d'autre part, des besoins variables exprimés par les utilisateurs. C'est alors autour de ce noyau qu'opère un ensemble de mécanismes essentiels et nécessaires pour la réalisation de la composition de services. Dans cette thèse, nous avons identifié cinq mécanismes principaux à savoir : la découverte de services, le matching de services, l'évaluation du contexte et de la qualité de service, la sélection de services, et le monitoring de services. Ces mécanismes sont en relation directe avec le processus de composition dynamique de services et ils sont intégrés et utilisés à des étapes différentes de ce processus. Dans la littérature, il existe plusieurs modèles qui utilisent la composition dynamique de services pour résoudre les problèmes associés aux environnements AmI. Chaque modèle fournit une partie de la réponse en se focalisant sur différents aspects de la composition dynamique de services dans les environnements à intelligence ambiante. Cependant, aucun des modèles étudiés n'apportent une réponse globale et complète aux problèmes rencontrés en environnements AmI. Les principales faiblesses communes à tous ces modèles sont les suivantes: manque d'expressivité du modèle de connaissances, faible degré d'automatisation du processus de composition de services, faible prise en compte en dynamique des événements, faiblesse en termes de gestion des incertitudes et d'auto-adaptation au contexte d'utilisation. C'est dans cette optique que s'inscrivent alors les travaux de recherche menés dans le cadre de cette thèse. Il s'agit de proposer un nouveau système AmI intégrant plusieurs contributions pour pallier aux différents inconvénients des modèles étudiés. Ces différentes contributions pavent des chemins orientés dans une même direction : améliorer la composition dynamique de services fournis par les dispositifs d'un environnement ambiant.

1.2. Contexte générale : intelligence ambiante

1.2.1. Informatique ubiquitaire

Le concept d'informatique ubiquitaire «*Ubiquitous Computing*» ou informatique ambiante «*Ambient Computing*» a été développé par Mark Weiser **[1]** au cours des années 80 à Xerox PARC, pour désigner sa vision de l'ordinateur du 21^{ième} siècle. Weiser constate que les technologies ancrées dans nos activités quotidiennes sont celles qui savent s'y fondre, jusqu'à disparaître. D'après sa vision, l'ordinateur sera intégré dans les objets de la vie courante et il deviendra invisible dans notre environnement de telle sorte qu'il rendrait continuellement des services indispensables sans que personne ne remarque sa présence. L'enjeu donc de cette évolution est de rendre l'informatique invisible jusqu'à n'apparaître qu'en arrière plan de notre conscience: « *A new way of thinking about computers in the world, one that takes into account the natural human environment and allows the computers themselves to vanish in the background* » **[1]**. Ainsi, l'informatique ubiquitaire (i.e. omniprésente) va au delà d'une

simple intégration de la technologie dans l'environnement, elle vise même à la rendre invisible. C'est la notion de disparition de la technologie dans l'environnement de la vie quotidienne, comme le souligne Weiser : «*The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it* » **[1]**. Selon Streitz et Nixon **[2]**, cette disparition comprend deux aspects : le premier concerne la disparition physique, tandis que le deuxième concerne la disparition mentale.

La disparition physique est le résultat des grandes innovations technologiques qui tendent vers la miniaturisation des dispositifs électroniques et informatiques. Il est d'ores et déjà devenu possible d'enfouir des systèmes électroniques complexes dans les objets courants. C'est objets sont munis d'une capacité de calcule et de communication et deviennent ainsi des équipements intelligents. Ces derniers vont des terminaux mobiles (PC portables, assistants PDA et PC de poche) à des équipements plus spécialisés, tels que les badges d'identification, les vêtements intelligents, les téléphones portables, les appareils photo numériques, les capteurs, les actionneurs ou tout autre équipement de contrôle/commande. La communication entre ces équipements intelligents est supportée par plusieurs standards de communication développés dans le cadre des réseaux sans fil. Grâce à cette forte présence des équipements intelligents et leurs capacités de communication à l'aide des réseaux sans fil, l'environnement devient intelligent et forme une interface naturelle avec le système d'informations.

La notion de disparition mentale de la technologie est, quant à elle, un aspect fondamental pour rendre la vie plus confortable à l'utilisateur. En effet, la croissance du nombre des équipements intelligents et leur diversité pose un sérieux souci pour l'utilisateur, concernant le temps et l'attention nécessaire qu'il doit consacrer pour interagir avec ces derniers. Il s 'agit dans cet aspect, de rendre les communications moins intrusives, utilisables d'une manière transparente à l'utilisateur. C'est ainsi que l'interaction homme - machine disparaisse au profit de l'interaction homme - information ou directement entres des personnes à travers un système de communication. On parle alors de la communication ambiante **[3]** qui exploite les possibilités de l'informatique ubiquitaire pour réduire cet environnement technologique à une interface de communication simple et intuitive.

Grâce à cette notion de disparition physique et mentale de la technologie, l'informatique se dispersera et se fondras dans l'environnement pour en devenir une partie intégrante et invisible. Cette orientation a donné naissance à plusieurs interprétations, qui constituent aujourd'hui autant de courants de l'informatique notamment: l'informatique diffuse (*Pervasive Computing*), l'informatique sensible au contexte (*Context-aware Computing*), et l'intelligence ambiante (*Ambient Intelligence*). L'informatique diffuse, est une initiative essentiellement industrielle marquée par l'implication d'IBM. L'accent est plutôt mis sur les aspects techniques avec notamment, le développement des supports matériels et logiciels nécessaires à la concrétisation de la vision de Weiser. L'informatique sensible au contexte constitue, quant à elle, l'un des développements majeurs de la vision de Weiser. Elle permette au système de réagir en fonction du contexte. Nous définirons plus précisément la notion de contexte au cours du **chapitre 3**. Enfin, l'intelligence ambiante est une nouvelle vision des systèmes intelligents offrant une multitude de fonctions et de services afin de répondre intelligemment à des besoins spécifiques des utilisateurs, à n'importe quel moment, et à

n'importe quel endroit. Cette nouvelle orientation de l'informatique sera développée dans la suite de ce chapitre.

1.2.2. Intelligence ambiante

Le terme intelligence ambiante (AmI*: Ambient Intelligence*) **[4, 5]** a été introduit pour la première fois en 1998 par la société Philips dans le cadre du projet «*Vision of the Future*». Dans ce projet, Philips mène une réflexion et une analyse prospective en interne sur l'évolution de l'électronique grand public. En 2001, l'ISTAG (*Information Societies Technology Advisory Group*) a publié un document regroupant un ensemble de quatre scénarii, illustrant ce que pourrait être un monde à "Intelligence Ambiante" à l'horizon de 2010 **[4]**. L'objectif de ce travail était d'une part, d'alimenter sur un long terme la recherche et d'autre part, d'évaluer les recherches européennes dans ce domaine émergeant.

Dans la littérature, il existe plusieurs définitions pour le terme intelligence ambiante. Dans ce qui suit, nous citons quelques unes sans être exhaustive. Selon l'ISTAG **[4]**, l'intelligence ambiante consiste à créer des environnements capables de prendre en compte les caractéristiques de chaque usager, de s'adapter et de répondre intelligemment à ses besoins spécifiques, d'agir de manière non intrusive et le plus souvent invisible, de permettre à l'usager d'accéder aux services de la façon la plus naturelle et intuitive possible, en exploitant la reconnaissance vocale, gestuelle ou la manipulation d'objets tangibles. Selon *Reignier* **[6]**, l'intelligence ambiante est un paradigme résultant de l'intersection de l'informatique ubiquitaire et de l'intelligence artificielle. L'intelligence ambiante trouve donc son origine dans la vision de l'informatique ubiquitaire dans laquelle les ordinateurs imprègnent l'environnement quotidien tout en étant transparents à l'utilisateur, mais en y ajoutant la notion d'intelligence, c'est-a-dire la faculté de perception et d'analyse de l'environnement, des utilisateurs et de leurs activités afin de réagir et s'adapter dynamiquement en fonction du contexte.

Nous constatons que ces différentes définitions sont assez similaires dans le fond. En effet, ces définitions s'accordent sur le fait que, à travers le paradigme de l'intelligence ambiante, l'objectif est de créer des environnements ou des écosystèmes intelligents permettant d'améliorer la qualité de vie des usagers. Dans de tels environnements (i.e. espaces), le monde réel et le monde virtuel se mélangent pour transformer les dispositifs et équipements de notre vie quotidienne en objets communicants offrant pro-activement des services d'assistance à valeur ajoutée afin de répondre aux besoins des usagers. Dans **[7]**, les auteurs considèrent qu'un environnement AmI est la fusion de deux tendances importantes : "informatique ubiquitaire" et "interfaces utilisateur". Ces deux dernières ont permis un dialogue direct naturel et intuitif entre les personnes et les applications/services (**Figure 1.1**).

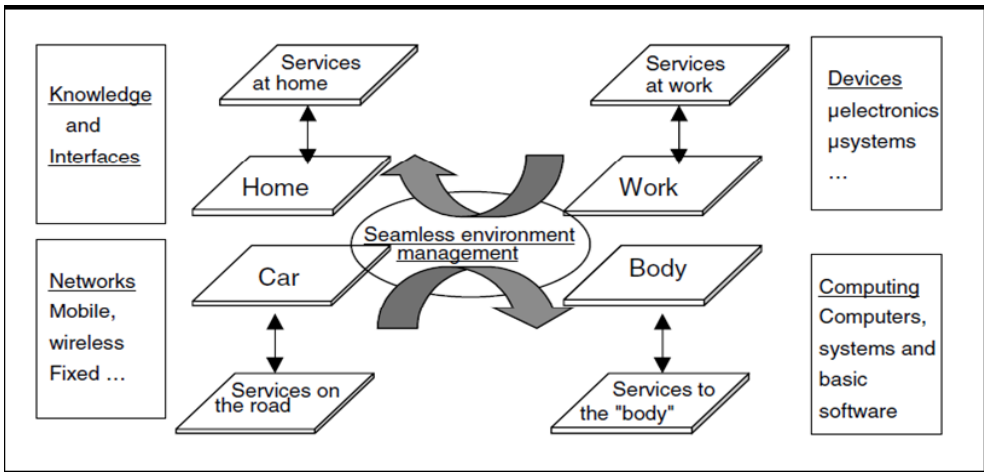


Figure 1.1: Espace AmI **[7]**.

Par ailleurs, le fonctionnement d'un environnement ambiant doit reposer sur un système capable d'assurer sa gestion d'une façon permanente et transparente. On parle dans ce cas de systèmes intelligents ambiants ou simplement systèmes AmI. Ces systèmes sont en mesure d'assurer un contrôle total des espaces intelligents en étant capables de :

- Percevoir les événements générés par des entités. Une entité peut être une personne, un équipement, un lieu ou une application informatique. Les entités sont en général caractérisées par des propriétés : un lieu peut être caractérisé par exemple non seulement par sa topologie mais également par ses niveaux de bruit, de luminosité, de température et d'humidité ;
- Collecter et interpréter les informations à partir de sources hétérogènes, interpréter l'intention de l'humain ;
- Réagir en conséquence de manière appropriée et transparente.

La particularité des systèmes AmI réside dans leur capacité à adapter continuellement et automatiquement la même fonction ou service (par exemple le rappel de rendez-vous), à différents contextes et modes d'usage (affichage d'un message textuel sur le PC de travail de l'usager si ce dernier est sur son lieu de travail, annonce d'un message vocal depuis le téléphone portable de l'usager si ce dernier est dans sa voiture et qu'il est en train de conduire, etc.). Ces systèmes possèdent aussi la propriété de découvrir dynamiquement les objets immatériels (services d'informations divers) ou physiques (robots, capteurs, actionneurs, équipements multimédia, etc.) disponibles dans l'environnement, puis de les exploiter pour fournir des services aux usagers.

Plusieurs types d'applications potentielles sont devenus envisageables dans le cadre des systèmes AmI pour différents espaces intelligents (e.g. maison, bureau, voiture, ville, etc.). Ainsi, dans le cadre des applications de maintien à domicile des personnes âgées ou dépendantes, un système à intelligence ambiante peut offrir une multitude de services réactifs ou proactifs permettant d'améliorer la qualité de vie et l'état physique, mental, et le bien-être social des usagers **[8]**. Ces services peuvent être de plusieurs types : assistance à la mobilité, assistance cognitive, sécurité, surveillance médicale, maintien du lien social, etc. Il est ainsi possible d'imaginer une multitude de services comme : Rappeler à l'usager de prendre ses

médicaments, envoyer une alarme au corps hospitalier ou aux proches en cas d'accident (chute, intoxication, électrocution, brûlure, complication médicale, etc.), etc. D'autres exemples de services d'assistance au quotidien et agissant sur le long terme peuvent être envisagés pour permettre à l'usager de maintenir une bonne hygiène de vie, afin de prévenir ou réduire les effets d'éventuelles maladies chroniques (coaching d'activités physiques, stimulation cognitives, etc.). D'autres types de services peuvent être conçus comme ceux agissant sur l'environnement pour répondre aux préférences et aux besoins des usagers : Fermer/ouvrir les volets, allumer/éteindre la lumière, régler le confort ambiant, chercher un objet égaré, etc.

1.2.3. Robotique ubiquitaire

Grâce à l'apparition des paradigmes de l'informatique ubiquitaire, de l'intelligence ambiante, et la disponibilité des technologies de communications sans fil (réseaux de capteurs, réseaux mobiles, smart phones, etc.), des technologies logicielles (Web services, middleware), d'identification (RFID, biométrie), etc., une nouvelle classe de robots de service est envisagée : le robot ubiquitaire. Cette classe de robots apparaît comme un domaine de recherche très prometteur **[9] [10].** Elle a fait l'objet de plusieurs travaux qui l'ont étudiée sous différents angles : les robots en réseaux (*network robot systems*) **[11]**, réseaux de capteurs-actionneurs (*sensor-actuator networks)* **[12]**, et la robotique ubiquitaire (*ubiquitous robotics*) **[9]**. Les enjeux autour de cette nouvelle forme de la robotique sont considérables avec un potentiel fort et des perspectives de croissance prometteuses, notamment sur le marché de la robotique de service. Des acteurs de l'industrie informatique de tout premier plan comme Google, Microsoft, Intel, etc. s'intéressent de plus en plus à ce marché.

Dans cette vision, un robot est considéré comme une entité douée de facultés de perception, de raisonnement, d'action et de communication, qui peut être recrutée dynamiquement dans une fédération de composants pour fournir des services. La notion de service telle qu'elle est conçue dans les systèmes d'information distribués peut être perçue de la même manière qu'en robotique ubiquitaire. En effet, les compétences d'un robot ubiquitaire et des objets environnants peuvent êtres publiées dynamiquement dans des annuaires sous forme de services permettant ainsi, à chaque objet de coordonner ses services avec ceux fournis par les autres objets de l'environnement. Dans ce cadre, le robot, en soi, n'est qu'une intelligence limitée. Ses actions sont commandées par les services en coordination avec d'autres entités de l'environnement. Son autonomie lui sert à maintenir son état de fonctionnement **[13].** Par conséquent, le modèle d'interaction entre le robot, l'environnement et l'homme est totalement repensé puisqu'il ne s'appuie plus sur un schéma pré établi ; l'environnement d'évolution étant considéré comme ouvert, partiellement connu et fortement dynamique. Un robot ubiquitaire doit ainsi être sémantiquement et automatiquement interopérable avec les objets (capteurs/actionneurs/robots) qu'il découvre dans l'environnement plutôt que d'être pré programmé statiquement pour cet environnement. Il doit être capable non seulement d'utiliser ses propres objets (capteurs et actionneurs) mais aussi d'interagir avec d'autres objets de l'environnement et d'adapter dynamiquement ses services face à des changements de contextes.

L'utilisation conjointe de la robotique ubiquitaire et de l'intelligence ambiante constitue un choix synergétique pour créer des écosystèmes exploitant des objets communicants (capteurs, actionneurs, terminaux numériques, artefacts intelligents, robots d'assistance, etc.) d'un environnement connecté. Ces écosystèmes constituent des espaces physiques et numériques riches et offrant une multitude de services intelligents réactifs et/ou proactifs, visant à améliorer nos cadres de vie. La gestion et le contrôle de ces écosystèmes nécessite la mise en place des systèmes AmI adéquats permettant d'intégrer les robots ubiquitaires dans des espaces intelligents (*smart spaces*) à petite ou large échelle (maisons intelligentes, bâtiments, espaces urbains), afin d'offrir, en tout lieu, à tout instant et de manière transparente des services à valeur ajoutée : Assistance cognitive, sécurité, confort, divertissement, etc.

1.3. Composition des environnements d'Intelligence Ambiante

Dans ce qui suit, nous dressons tout d'abord un panorama des technologies actuelles permettant de mettre en œuvre des systèmes à intelligence ambiante.

1.3.1. Artefacts intelligents

Un artefact intelligent correspond à n'importe quel objet physique fonctionnel de la vie quotidienne associant capteurs, unité de traitement, unité de communication et mémoire. Il est capable de percevoir son environnement, communiquer avec d'autres artefacts et éventuellement réagir selon une base de règles définie a priori. Á travers sa capacité d'interaction directe avec un humain, un artefact intelligent peut assister une personne dans ses tâches quotidiennes en lui offrant des modes d'interaction intuitifs **[14]**. Contrairement aux dispositifs médicaux mobiles, les fonctionnalités médicales supplémentaires des artefacts intelligents ne sont généralement pas visibles de l'extérieur. Dans ce domaine, on peut citer l'exemple de Smart Pillow, un oreiller intelligent développé par la société Philips. Ce système surveille les paramètres vitaux de l'utilisateur, tels que la température, la respiration, le pouls, et en cas d'urgence ou de maladie, avise le personnel médical **[15]**. Dans le domaine de l'assistance cognitive, on peut citer les travaux menés au laboratoire DOMUS de l'université de Sherbrooke-Canada. Le domicile est considéré comme une prothèse cognitive capable d'assister une personne ayant des déficits cognitifs (problèmes d'attention, de mémoire, de planification, etc.), par exemple en lui rappelant les tâches à réaliser, ou en l'aidant à gérer son temps ou à se préparer pour des rendez-vous **[16]**. Nous pouvons citer également les travaux de l'équipe STARS de l'INRIA sur l'analyse des comportements des personnes atteintes de la maladie d'Alzheimer **[17]**.

Le concept d'artefacts intelligents a été étudié et développé dans des applications autres que les médicales. Smart Sofa est un canapé instrumenté qui a été développé par le Trinity College de Dublin-Irelande. Il permet d'identifier les personnes assises dessus et de fournir des services personnalisés basés sur ces informations **[18]**. Les ustensiles intelligents de cuisine sont des exemples d'artefacts mis en œuvre au Massachusetts Institute of Technology **[19]** : (i) une casserole, équipée d'une puce, qui indique si elle est trop chaude pour être manipulée ; (ii) une cuillère qui fournit des informations sur la température et la viscosité de la nourriture ; (iii) une bouilloire qui informe l'utilisateur du temps d'attente pour la préparation de son thé **[20]**. D'autres prototypes d'artefacts intelligents ont été développés

dans la même logique comme la tasse de café qui communique le type de café et la température du liquide qu'elle contient ; ou bien encore, la nappe interactive qui permet de saisir une commande dans un restaurant **[21]**.

1.3.2. Accessoires et Vêtements Intelligents

Concernant les technologies mobiles pour la santé et l'autonomie, deux grands courants de la recherche sont devenus prédominants au cours des dernières années : Les accessoires intelligents et les vêtements intelligents.

Concernant la première catégorie, l'exemple le plus marquant est le projet Google Glass, une paire de lunettes intelligente qui offre des services de communication et de navigation à l'usager mobile. Starner et al. **[22]** ont développé Gesture Pendant, un pendentif qui reconnaît des gestes prédéfinis de l'usager et exécute des actions de contrôle correspondantes. Kikin-Gil et al. ont développé des accessoires intelligents pour permettre la communication non-verbale au sein de petits groupes d'adolescents **[23]**. Les montres ou bracelets sont des accessoires populaires qui sont de plus en plus utilisées dans des applications de surveillance médicale. Plusieurs modèles de montres intelligentes sont déjà disponibles dans le commerce à l'image de l'Actiwatch de la société Cambridge Technology. Cette montre est équipée d'un accéléromètre miniature qui mesure l'activité physique de son porteur. La combinaison de ce capteur avec d'autres capteurs a permit d'étendre les fonctionnalités de la montre à la surveillance des troubles du sommeil : insomnie, humeur, dépense d'énergie, mouvements périodiques des membres pendant le sommeil **[24]**. Il existe d'autres exemples de montres-bracelets offrant d'autres types de fonctionnalités, telles que la détection de chute, le test de fibrose kystique **[25]**, la mesure de glycémie **[26]**, la surveillance de l'oxygénation du sang **[27]**, l'envoi d'appels d'urgence **[28]**.

Le concept de vêtement intelligent (wearable computing- informatique à porter) repose sur l'idée qui consiste à avoir des ordinateurs miniatures comme partie intégrante des vêtements qui nous habillent ou des accessoires que nous portons. Les vêtements intelligents sont des artefacts portables destinés à accompagner l'utilisateur dans ses déplacements. Dans cette catégorie, on peut citer le projet Smart Shirt développé à Georgia Institute of Technology **[29]**, où différents types de capteurs ont été intégrés dans la conception d'une chemise intelligente, pour permettre le suivi de paramètres vitaux comme la fréquence cardiaque, l'électrocardiogramme (ECG), la respiration, la température, etc. **[30]** et **[31]**. Un autre exemple de système portable de surveillance de paramètres physiologiques est la veste intelligente (Smart Vest) développée par Pandian et al. **[32]**. Le système permet de surveiller des paramètres physiologiques tels que l'ECG, la pression artérielle, la température du corps et la fréquence cardiaque. Ces paramètres ainsi que la géo-localisation du porteur sont transmis aux stations de surveillance **[33]**. Certains systèmes sont déjà commercialisés tels que le gilet LifeShirt de la société VivoMetrics. Ce système permet d'effectuer de manière non-invasive une pléthysmographie respiratoire inductive pour surveiller des paramètres cardiorespiratoires (mesure des variations de volume du thorax et de l'abdomen dues aux mouvements respiratoires) **[34]**. Actibelt, est une ceinture intelligente qui est équipée d'un accéléromètre tridimensionnel intégré dans sa boucle. Ce capteur permet d'analyser l'activité physique du porteur sur une longue période **[35]**. On peut trouver dans la littérature d'autres

exemples de vêtements intelligents : protections actives de la hanche **[36]**, chaussures intelligentes **[37],** genouillères intelligentes **[38]**.

1.3.3. Identification

Parmi les systèmes embarqués capables d'identifier ou de suivre une personne ou un objet, on peut citer l'identification à base de radiofréquences appelée communément RFID (Radio Frequency IDentification). Cette technologie, actuellement mature, connaît une véritable explosion, en particulier grâce aux efforts multiples et continus de miniaturisation des circuits intégrés. Son principe consiste en des radio-étiquettes (tags) RFID qui sont capables d'émettre un signal contenant un identifiant en réponse à une requête envoyée par un lecteur. On distingue deux types d'étiquettes : Les étiquettes passives et les étiquettes actives. Les étiquettes passives (sans batterie ou pile) sont alimentées par le champ électromagnétique généré par le lecteur à l'émission d'une requête. Les étiquettes actives, quant à elles, sont alimentées par une batterie ou une pile. Outre l'identification, les étiquettes actives peuvent potentiellement êtres utilisées comme télémètres basés sur la mesure de la puissance du signal Radiofréquence. On dénombre quatre bandes principales de fréquences dédiées pour les applications de la technologie RFID : Les basses fréquences, aux alentours de 125 kHz, les hautes fréquences à 13,56 MHz, la bande UHF, entre 800 et 900 MHz et enfin les hyperfréquences avec des fréquences à 2,45 GHz et 5,8 GHz.

a) Grille d'étiquettes RFID b) Robot Centribot

Figure 1.2 : Instrumentation du sol d'un appartement par des tags RFID passives **[39]**

La **figure 1.2** illustre l'instrumentation d'un appartement dans le cadre du projet PEIS Ecology mené par l'université D'Orebro-Suède. Le système utilise une grille de tags RFID passifs sous le parquet pour guider le robot Centribot. Ce dernier est équipé d'un lecteur de tags RFID, dont l'antenne s'étend depuis la base.

1.3.4. Robots de services

L'utilisation de robots de services pour l'assistance aux personnes dans l'exécution de leurs tâches ou activités quotidiennes est toujours en phase d'expérimentation et de recherche de modèles économiques viables. Dans ce qui suit, nous nous limiterons aux deux catégories suivantes : Les robots d'assistance à la mobilité ou au déplacement et les robots d'assistance domestique.

1.3.4.1. Robots d'assistance à la mobilité

Ces robots sont en général conçus pour compenser les déficiences motrices de leurs utilisateurs et aider ces derniers dans leurs activités physiques quotidiennes comme se lever/s'asseoir, marcher, monter les escaliers, etc. Dans cette catégorie de robots, on trouve les robots fauteuils, les robots cannes, les robots déambulateurs et les robots exosquelettes. La **figure 1.3** illustre quelques prototypes de robots : (a) le concept de fauteuil roulant robotisé CARRIER développé par l'Université des arts appliqués de l'Industrial Design Studio 2 Esslinger en Autriche. Ce robot fait également office de verticalisateur ; (b) le fauteuil roulant robotisé Sharioto de l'Université de Louvain **[40]** ; (c) La cane intelligente iCane développée à l'université de Nagoya **[41]** ; (d) le déambulateur RobuWalker développé par la société Robosoft et l'ISIR pour l'aide à la verticalisation et au déplacement; (e) l'exosquelette Hal de la société Cyderdyne; (f) l'exosquelette EiCOSI, pour l'assistance aux mouvements de l'articulation du genou, développé au laboratoire LISSI **[42]**. Les exosquelettes ou orthèses sont des dispositifs mécatroniques que l'on qualifie de robots portables. Ils sont utilisés dans le but d'augmenter, d'assister ou de restaurer les mouvements des personnes dépendantes.

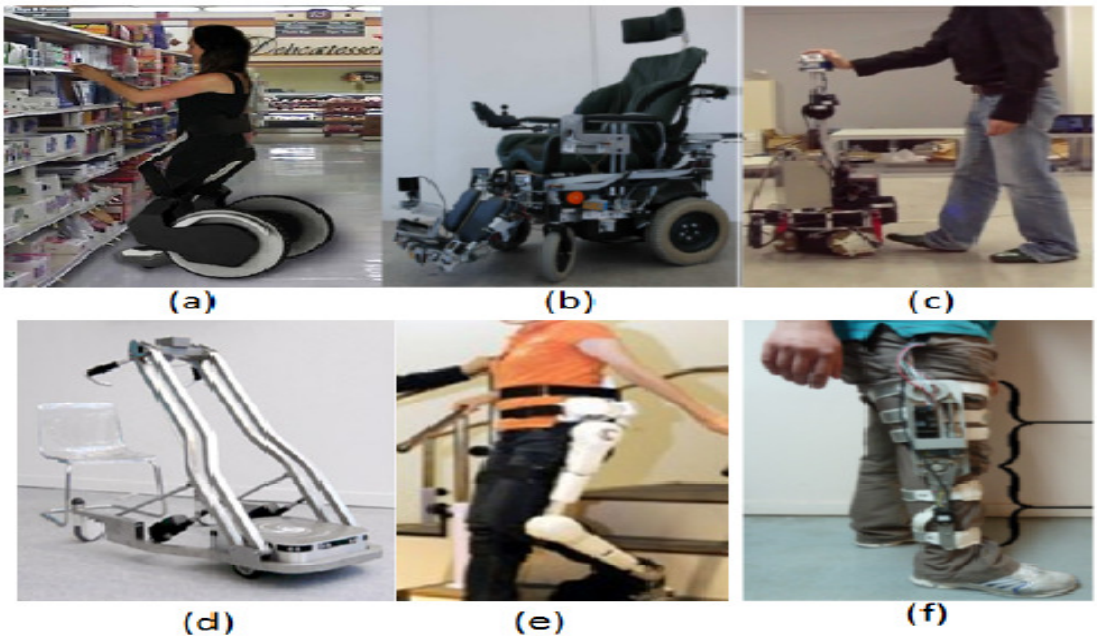


Figure 1.3 : Robots d'assistance à la mobilité : (a) CARRIER (b) Sharioto (c) iCane (d) RobuWalker (e) Hal (f) EiCOSI.

1.3.4.2. Robots d'assistance domestique

Dans la catégorie des robots d'assistance domestique qu'on appelle aussi robots personnels, on peut citer les robots d'assistance relationnelle qui regroupent les robots compagnons et les robots pour l'éveil sensoriel. Les robots compagnons ont une fonction d'assistance cognitive auprès de personnes isolées et fragilisées et peuvent communiquer avec la personne en tant qu'objets mobiles et communicants.

Les robots compagnons sont en général constitués d'une base mobile avec une tête. Ils sont munis de capteurs, de reconnaissance et de synthèse vocale, d'interfaces de connexion à Internet et de fonctions de navigation autonome. Les fonctions principales pouvant être prises en charge par un robot compagnon concernent essentiellement l'assistance cognitive, allant

des aide-mémoire aux stimulations (exercices physiques et intellectuels), ainsi que les fonctions basiques de communication comme le courrier électronique ou l'interaction social (accès aux réseaux sociaux, famille, amis, etc.). La **figure 1.4** illustre quelques exemples de robots personnels : Kompai de Robosoft, Ava d'iRobot, PR2 de Willow Garage.

Figure 1.4 : Robots personnels : (a) Kompai (b) Ava (c) PR2.

Les robots d'éveil sensoriel sont généralement conçus pour favoriser l'interaction et la communication par le contact direct avec la personne. Ils sont destinés aux personnes souffrant de troubles cognitifs tels que l'autisme, les maladies d'Alzheimer, etc. Ils se présentent sous la forme d'animaux de compagnie robotisés à l'image du robot chien Aibo de Sony. Ces robots sont équipés de capteurs leur permettant de réagir au toucher et à la voix et peuvent émettre des sons. Ils sont dotés d'une intelligence leur permettant essentiellement une communication et une interaction affective, personnalisée avec l'individu.

Le robot PR2 de Willow Garage est sans doute le robot personnel le plus abouti. Il dispose d'une base mobile à roues et de deux bras qui possèdent chacun 7 degrés de liberté. Ce robot reste néanmoins un outil d'expérimentation pour la recherche. La robotique de service multitâche pour des applications domestiques est toujours au stade de l'expérimentation et de la recherche, comme en témoignent les nombreux travaux, notamment sur les robots humanoïdes. A ce jour, il n'existe pas encore de produits commercialisés dans le domaine grand public.

1.4. Caractéristiques et défis des environnements à intelligence ambiante

Les applications de l'intelligence ambiante consistent en un énorme réseau d'entités physiques (robots, capteurs, actionneurs, équipements multimédia, etc.) et virtuelles (services d'informations divers, composants logiciels, etc.) hétérogènes interconnectés et qui interagissent avec l'humain. Ces applications évoluent dans des environnements fortement contraints : hétérogènes, ouverts, dynamiques, incertains et multi-échelles. Ces caractéristiques imposent ainsi des défis scientifiques considérables pour le développement, le déploiement et la maintenance de ces applications et de leurs supports d'exécution. Afin de répondre à ces défis, un système intelligent ambiant doit remplir certaines exigences

techniques. Tout d'abord, un tel système doit assurer l'interopérabilité et un haut niveau d'abstraction de l'hétérogénéité des entités de l'environnement ambiant. Ensuite, un système AmI doit être construit autour de trois aspects fondamentaux à savoir : l'intelligence, la sensibilité au contexte et l'auto-adaptation. Ces aspects essentiels procurent au système AmI la capacité de raisonnement pour la reconnaissance de contextes et la prise de décision, l'adaptation et l'extensibilité afin d'assurer la pérennité, l'évolutivité du système par rapport à l'évolution de l'environnement ambiant et des usagers. Par ailleurs, ces trois aspects doivent être réalisés d'une manière invisible et transparente aux utilisateurs tout en assurant la sécurité et la protection de leurs vies privés. Enfin, un système AmI doit assurer un passage à l'échelle (scalabilité) des différents mécanismes utilisés afin de répondre efficacement aux attentes des utilisateurs dans les environnements à large échelle. Les caractéristiques des environnements ambiants ainsi que les défis imposés aux systèmes AmI sont illustrés sur la **figure 1.5**. Sur cette figure, les caractéristiques des environnements AmI sont marquées en traits discontinus tandis que les défis des systèmes AmI sont marqués en traits continus.

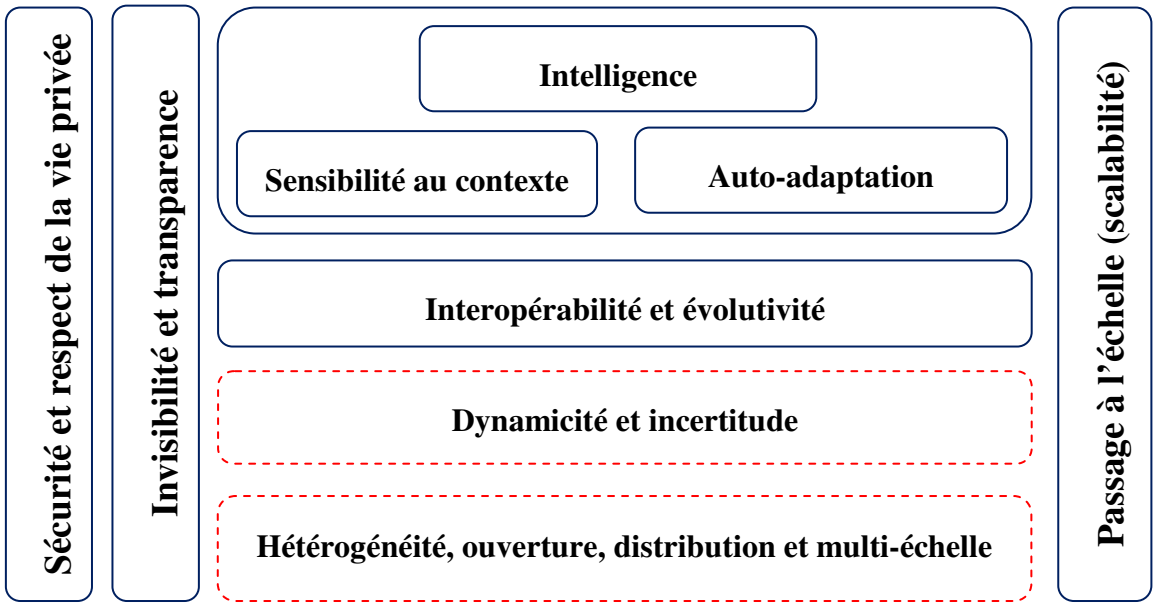


Figure 1.5 : Caractéristiques et défis des environnements à intelligence ambiante

1.4.1. Hétérogénéité, Ouverture, Distribution et Multi-échelle

L'hétérogénéité est l'une des caractéristiques majeures des environnements ambiants. Cette hétérogénéité est due, d'une part, aux diverses technologies des dispositifs intégrés dans les environnements ambiants, telles que les technologies des capteurs et des actionneurs embarqués, d'autre part, à la diversité des technologies de communication filaires et sans fil. En effet, les entités d'un environnement ambiant, souvent issues de l'électronique grand public, sont très variés (ordinateur portable, PDA, smart phone, etc.) et apparaissent avec leurs protocoles de communication standards ou propriétaires, des langages de programmation distincts et des modèles de données sémantiquement différents. Ces modèles qui reposent sur des caractéristiques propriétaires et peu interopérables, posent de grandes difficultés pour le partage des données et l'interaction entre les entités distincts. Par ailleurs, les dispositifs fonctionnent sous des systèmes d'exploitation distincts et éventuellement avec des

configurations matérielles et logicielles différentes. Ainsi, l'hétérogénéité se manifeste sur trois niveaux essentiels: niveau architectures matérielles (entités de calcul), niveau infrastructure de communication (protocoles et technologies réseaux) et niveau composants logiciels (système d'exploitation, langage de programmation et modèle de données). L'ouverture de l'environnement à des entités électroniques distinctes et variées dans un contexte évoluant rapidement est aussi une caractéristique importante des environnements ambiants. Ces entités, considérées comme des ressources, ne se limitent pas à un seul endroit mais sont souvent distribuées dans plusieurs lieux. Elles sont réparties et interconnectées via des réseaux filaires et/ou sans fil. En plus à ces aspects d'hétérogénéité, d'ouverture et de distribution, les environnements ambiants se caractérisent par un nombre important de ressources (de calcul, de communication et d'interaction, de capteurs et d'actionneurs) à gérer.

1.4.2. Dynamicité et incertitude

Les environnements ambiants sont également très dynamiques et incertains. La nature dynamique est due aux changements fréquents qui se produisent dans les environnements ambiants alors que la nature incertaine est due à l'apparition aléatoire de ces changements. En effet, le changement constant de l'environnement se traduit par la mobilité de l'humain (i.e. sa localisation peut changer en permanence), l'apparition et la disparition d'équipements comme conséquences aux interactions entre l'humain et les équipements électroniques/physiques **[43]**. Par exemple, la présence d'un téléphone portable ou d'un assistant électronique en un lieu n'est pas permanente. En particulier, la mobilité des personnes et des objets (capteurs, actionneurs, robots, etc.) est une caractéristique intrinsèque qui peut influencer considérablement la qualité des services fournis, et entraîner dans la plupart des cas une disparition temporelle de ces services. Par conséquent, de nouvelles fonctionnalités seront offertes par les dispositifs qui apparaissent et d'autres fonctionnalités seront supprimées à cause des dispositifs qui disparaissent. Les changements de l'environnement se traduisent également par l'évolution régulière des ressources disponibles et des besoins des utilisateurs. Par exemple, les réseaux sont mis à jour régulièrement et les besoins des utilisateurs peuvent évoluer en fonction de leurs activités. D'une façon générale, le changement constant de l'environnement ambiant se traduit par le changement perpétuel du contexte d'utilisation. Toutefois, ce changement de contexte peut atteindre parfois des cas plus extrêmes là où des pannes matérielles et logicielles surviennent. Plusieurs raisons peuvent être à l'origine de ces pannes telles que: les défaillances physiques ou défauts d'énergie des dispositifs, les déconnexions réseaux, les arrêts lors de la découverte de services, les arrêts (échecs) lors d'invocation et d'exécution des services. Ces pannes fréquentes surgissent souvent d'une façon aléatoire et peuvent causer la dégradation de la qualité du service fourni qui peut aller d'une simple indisponibilité momentanée du service jusqu'à un arrêt total de son fonctionnement. En outre, les informations sur le contexte acquis peuvent devenir ambiguës, incomplètes et parfois même incohérentes en raison des défaillances des capteurs.

En conclusion, dans un environnement ambiant, les services et les dispositifs ont souvent des comportements imprévisibles et peuvent basculer dans la plupart des cas à des états imprévisibles qui sont interprétés comme une dégradation de la qualité fournie par le système. En conséquence, un service dans un schéma donné peut devenir indisponible et doit donc être remplacé afin de garantir la continuité de service. De plus, les ressources (dispositifs) qui

fournissent ces services peuvent être remplacées avant qu'elles ne deviennent physiquement hors de service. Ces aspects sont pris en compte par les mécanismes d'auto-adaptation.

1.4.3. Interopérabilité et évolutivité

La mise en œuvre de techniques d'interopérabilité permet de traiter principalement les problèmes liés à l'hétérogénéité des environnements à intelligence ambiante. L'hétérogénéité pose le problème de partage des données et indirectement des connaissances contextuelles. Elle concerne d'une part, l'incompatibilité des moyens de communication (couches protocolaires), et d'autre part, les différents modèles représentant les données. L'émergence des middlewares (intergiciels) a permis de cacher l'hétérogénéité à travers un support de communication interopérable permettant de simplifier l'intégration et une perception abstraite des différentes entités de l'environnement ambiant. Ces middlewares exploitent des techniques intergicielles développées dans le cadre des systèmes distribués telles que CORBA, RMI, SOA, etc. Les intergiciels permettent de simplifier la mise en œuvre des systèmes répartis tout en masquant l'hétérogénéité des différents équipements et réseaux utilisés. En effet, les services d'intelligence ambiante interagissent avec les capteurs et les actionneurs à travers des intergiciels qui offrent un support de communication interopérable permettant de simplifier l'intégration et la perception abstraite des entités de l'environnement. Plusieurs intergiciels et protocoles d'interaction ont été standardisés pour le domaine de l'informatique ubiquitaire et l'intelligence ambiante notamment dans le cadre des architectures SOA pour dispositifs telles qu'UPnP et DPWS. Ces derniers reposent sur l'usage des protocoles de l'internet comme infrastructure de communication. Par ailleurs, afin de garantir la pérennité des services d'intelligence ambiante et l'évolution de tout l'environnement d'intelligence ambiante pour accompagner les besoins des usagers, les intergiciels doivent disposer de fonctions d'adaptation prenant en compte la dynamicité de l'environnement en termes d'apparition ou disparition de services ou d'équipements. Enfin, pour faciliter la mise en œuvre de nouvelles applications, les intergiciels doivent fournir des outils qui permettent de simplifier la mise en œuvre des services, en utilisant par exemple des techniques d'assemblage de composants logiciels ou des techniques de composition de services. Ces techniques seront développées dans les prochains chapitres.

1.4.4. Auto-adaptation

Face à la nature dynamique et incertaine de l'environnement ambiant, des mécanismes d'auto-adaptation sont alors nécessaires afin d'assurer une continuité de service et répondre aux attentes des utilisateurs en fonction de leurs contexte, entre autre, leur activité, leur localisation et les équipements dont ils disposent. De plus, l'adéquation entre ressources requises et fournies doit être établie et rétablie dynamiquement selon l'évolution des entités disponibles. L'intégration des mécanismes d'auto-adaptation dans un système intelligent constitue la clé pour gérer les changements de contexte et permettent d'augmenter le degré d'autonomie, de robustesse et de flexibilité d'un tel système. L'autonomie reflète la capacité du système à s'adapter seul aux changements sans aucune intervention humaine, alors que la robustesse reflète la capacité du système à assurer la continuité du service en cas de pannes et à s'adapter constamment au contexte d'utilisation. La flexibilité, quant à elle, reflète la souplesse du système dans la façon d'opérer l'adaptation qui doit rester complètement

transparente à l'utilisateur. Concrètement, une auto-adaptation peut être réalisée par trois mécanismes essentiels: autoréparation, auto-configuration et auto-apprentissage. L'autoréparation consiste à réparer la partie défectueuse fournie par le système soit par la maintenance de cette dernière ou bien par son remplacement dans le cas extrême. L'auto-configuration consiste à ajuster partiellement ou totalement la configuration fournie par le système soit pour remédier à une partie défectueuse ou bien pour s'adapter au nouveau contexte d'utilisation. Enfin, l'auto-apprentissage consiste à apprendre le processus des changements qui s'opèrent sur l'environnement ambiant. L'apprentissage se base sur l'historique des changements afin de prédire les changements futurs. Cet aspect permet au système d'anticiper les changements afin de mieux s'adapter aux nouvelles situations.

1.4.5. Sensibilité au contexte

Un système intelligent doit percevoir le contexte de l'environnement pour interagir plus « naturellement » avec l'utilisateur. La perception du contexte est réalisée par des capteurs de l'environnement physique. Toute sorte de capteurs est nécessaire à cette fin : caméras, microphones, capteurs biométriques, puces et lecteurs à radiofréquence (RFID) pour l'identification, etc. Les informations contextuelles concernent notamment l'environnement (température, humidité, etc.), l'utilisateur (la température corporelle, le rythme cardiaque, etc.), les appareils (affichage, énergie, etc.) et les applications (type, configuration, etc.). Les données du contexte sont en général perçues dans leurs formes brutes. Des mécanismes de description sémantique du contexte sont alors nécessaires afin de le rendre compréhensible et intelligible par le système. Ces mécanismes se basent essentiellement sur des méta-données décrites dans des langages sémantiques avec un haut niveau d'abstraction des connaissances. Un système est dits sensible au contexte «*context-aware* » s'il est capables de « sentir » en permanence le contexte courant et d'en tenir compte pour mieux répondre aux besoins et aux attentes des utilisateurs. Le contexte à prendre en compte est toujours différent et dépend du but de l'application que l'on veut rendre sensible. Dans différents contextes, les utilisateurs accèdent aux même données et aux mêmes services mais reçoivent des réponses qui peuvent être différentes ou présentées différemment ou encore à des niveaux de détails différents **[44]**. Par exemple, un médecin consulte le dossier médical d'un patient à l'hôpital à l'aide d'un ordinateur de bureau. Dans un autre contexte, il consulte le même dossier mais chez le patient à l'aide d'un ordinateur de poche. Il peut aussi avoir une synthèse vocale du même dossier dans un autre contexte.

1.4.6. Intelligence

Percevoir le contexte courant d'utilisation ne suffit pas, il faut pouvoir utiliser efficacement les informations du contexte perçu afin d'atteindre les objectifs assignés au système. L'intelligence « *smartness* » du système réside dans sa capacité d'analyser et d'agréger le contexte élémentaire dans un contexte plus complexe non perceptible directement par la sensibilité au contexte. En passant par plusieurs niveaux d'agrégation, le contexte élémentaire permettra d'inférer de nouvelles situations. Ainsi, le système peut évaluer une situation courante ou possible, comparer deux situations et juger de la meilleure. Pour ce faire, des mécanismes de raisonnement et de pro-action « *all the time everywhere* » sont utilisés afin d'accroître l'intelligence du système et sa capacité à rester attentif pour répondre à tout

moment et en tous lieux aux besoins des utilisateurs. Le système est proactif s'il est capable de suggérer et de proposer des actions correctives à l'utilisateur en fonction du contexte présent ou prédit et s'il est capable de prévoir un événement ou une situation et prendre les décisions adéquates. La prise de décision est en général suivie par l'exécution d'actions (services pro-actifs) comme par exemple :(i) ajuster la température et l'aération en utilisant les actionneurs disponibles dans l'habitation (ii) notifier à l'usager des commentaires, des suggestions ou des alertes instantanées ou en différé, à l'aide d'interfaces multimodales personnalisées.

1.4.7. Passage à l'échelle

Compte tenu de la nature ouverte et distribuée des environnements ambiants, un système intelligent est appelé à gérer des volumes de plus en plus grands d'entités connectées (utilisateurs, applications, appareils). Afin de garder des performances acceptables dans de telles conditions, le système intelligent doit être doté de mécanismes lui permettant d'assurer un passage à l'échelle (Scalabilité). Pour ce faire, le cœur d'un système intelligent doit être développé indépendamment du volume des utilisateurs et des appareils. De plus, certaines techniques d'adaptation peuvent être intégrées dans le système pour pouvoir répondre à chaque cas d'utilisation.

1.4.8. Invisibilité et transparence

Un environnement ambiant doit fournir une connectivité globale et s'affranchir de sa nature informatique pour ne plus laisser transparaître qu'un ensemble de fonctionnalités accessibles de manière intuitive. Un des premiers objectifs des environnements ambiants est donc de généraliser et banaliser les réseaux de communications sans fil et de rendre toute la technologie sous-jacente invisible à l'utilisateur. Un deuxième objectif est de lui rendre l'accès à l'information plus transparent vis à vis du lieu et du contexte actuel. Pour atteindre ces objectifs, les systèmes intelligents qui gèrent ces environnements ambiants doivent nécessiter un minimum d'intervention humaine. Cette intervention peut être réduite à la configuration initiale du système. Par la suite, ce système doit pouvoir s'adapter seul aux changements en s'appuyant, par exemple, sur des mécanismes d'auto-apprentissage afin de rester discret, transparents et invisible à l'utilisateur.

1.4.9. Sécurité et respect de la vie privée

La sécurité des systèmes informatiques et la protection de la confidentialité et de la vie privée des usagers sont deux caractéristiques importantes que doivent posséder les services d'intelligence ambiante. Les environnements d'intelligence ambiante peuvent comporter de multiples sources d'informations dont il faut contrôler et protéger l'accès tout en respectant les aspects éthiques et juridiques. Mettre en place des mécanismes fiables de contrôle d'accès aux services, et des systèmes de chiffrement et déchiffrement des données de la vie privée, sont des problématiques difficiles et complexes qu'il faut impérativement aborder pour permettre à des personnes dépendantes d'évoluer et d'interagir de manière sécurisée avec les services de l'intelligence ambiante sans atteinte à leur vie privée.

1.5. Visions et contributions de la thèse

Cette thèse s'inscrit dans le domaine de l'informatique ambiante et de la robotique ubiquitaire. Son objectif consiste à proposer un système AmI capable de répondre aux différents défis posés par les environnements à intelligence ambiante. Ces environnements se caractérisent notamment par l'ouverture, l'hétérogénéité, l'incertitude et la dynamicité des entités qui les constituent. Ces caractéristiques imposent alors des défis scientifiques importants pour la mise en œuvre de systèmes intelligents ambiants. Les principaux défis soulevés sont: l'intelligence, la sensibilité au contexte, l'auto-adaptation, l'interopérabilité, la transparence et le passage à l'échelle. Dans cette thèse, nous considérons que l'approche orientée service demeure la base de la construction d'un système AmI en raison de l'abstraction apportée dans la représentation des entités de l'environnement ambiant et du couplage lâche (faible) qu'elle permet entre ces différentes représentations. En effet, les entités hétérogènes ne sont représentées que par leurs fonctionnalités encapsulées dans la notion de service qui est indépendante des technologies d'implémentation. Ainsi, le paradigme orienté service garantit l'interopérabilité et le couplage faible nécessaires à la construction dynamique d'applications dans un environnement ouvert et hétérogène.

Cependant, afin d'appliquer correctement l'approche orientée service dans le cadre des systèmes intelligents ambiants, il est nécessaire d'identifier des mécanismes adéquats en orientation service permettant de réaliser les différents défis soulevés par les systèmes AmI. La composition de services est le mécanisme responsable de la construction dynamique d'applications en tenant compte, d'une part, des fonctionnalités offertes par l'ensemble des services disponibles et du contexte d'utilisation et d'autre part, des besoins variables exprimés par les utilisateurs. Ainsi, la composition de services constitue le noyau d'un système AmI réalisé par une approche orientée services. C'est autour de ce noyau qu'opère un ensemble de mécanismes essentiels et nécessaires pour la réalisation de la composition de services. Dans cette thèse, nous avons identifié cinq mécanismes principaux à savoir : la découverte de services, le matching de services, l'évaluation du contexte et de la qualité de service, la sélection de services, et le monitoring de services. Ces cinq mécanismes doivent reposer sur un modèle des connaissances qui modélise toutes les connaissances manipulées dans un environnement ambiant, et ce, dans des formats intelligibles et exploitables. Par ailleurs, ces différents mécanismes doivent être structurés dans une architecture générale qui détermine leurs relations et le fonctionnement global du système AmI. Une telle architecture doit aussi reposer sur un intergiciel permettant la communication entre les différentes entités de l'environnement. Enfin, tous les mécanismes d'un système AmI doivent être évalués sur des scénarii réels afin de montrer leurs performances et leurs faisabilités dans un environnement ambiant réel. En résumé, la mise en œuvre d'un système AmI dans un cadre orienté service se scinde en dix facettes essentielles à savoir : *architecture générale, modèle des connaissances, modèle d'évaluation de la QoS et du contexte, modèle de matching des services, modèle de sélection des services, modèle de découverte des services, modèle de composition des services*, *modèle de monitoring des services*, *modèle de communication* et *modèle d'évaluation*.

Afin de concrétiser notre vision et mener à bien notre travail, nous avons d'abord identifié onze principaux middlewares permettant la composition de services dans des environnements ambiants : *COCOA-PERSE (COnversation-based service Composition in pervAsive computing environments - Pervasive Semantic-aware Middleware), PEIS-Ecologie* (*Physically Embedded Intelligent System), URC-SURF (Ubiquitous Robot Compagnon-Service-oriented Ubiquitous Robotic Framework), MySIM* (*Spontaneous Service Integration for Pervasive Environment)*, *MEDUSA, MUSIC* (*Middleware Support for Self-Adaptation in Ubiquitous and Service-Oriented Environments), SeSCo* (*Seamless Service Composition)*, *SeGSeC* (*Semantic Graph-Based Service Composition*), *URS* (*Ubiquitous Robotic Space)*, *VRESCo* (*Vienna Runtime Environment for Service-oriented Computing*), *et WComp* (*Middleware for Ubiquitous Computing)*. Par la suite, nous avons étudié et analysé ces différents middlewares suivant les dix facettes citées précédemment pour un système AmI. Enfin, nous avons effectué une comparaison entre ces middlewares en analysant les points forts et points faibles de chaque middleware. Les principales faiblesses communes à tous ces middlewares sont les suivantes: manque d'expressivité du modèle de connaissances, faible degré d'automatisation du processus de composition de services, faible prise en compte en dynamique des événements, faiblesse en termes de gestion des incertitudes et d'auto-adaptation au contexte d'utilisation. A partir de cette étude comparative, nous avons proposé, dans cette thèse, un nouveau système AmI intégrant plusieurs contributions pour pallier aux différentes lacunes et faiblesses des middlewares étudiés. Ces différentes contributions pavent des chemins orientés dans une même direction : améliorer la composition de services fournis par les dispositifs d'un environnement ambiant.

La première contribution a porté sur la proposition d'un modèle des connaissances. Ce modèle est riche en termes d'informations car il englobe l'ensemble des connaissances manipulées dans un environnement ambiant. Ces informations concernent quatre entités principales à savoir : les services, les événements, le contexte et les utilisateurs. Tout d'abord, les services reflètent les fonctionnalités fournies par les différents dispositifs de l'environnement ambiant. Ainsi, nous distinguons les services concrets qui implémentent réellement ces fonctionnalités des services abstraits qui représentent des classes de services concrets. L'abstraction des services permet un haut niveau de raisonnement sur les services indépendamment de leurs implémentations concrètes. La qualité de service est également prise en compte en considérant aussi bien ses paramètres statiques que dynamiques. Les événements sont modélisés comme des déclencheurs pour le système. En effet, à chaque type d'événement est associée une règle événementielle. Cette dernière est une requête implicite qui permet de guider le système vers les objectifs souhaités par l'utilisateur en cas d'apparition d'un éventuel événement. Le contexte est modélisé, quant à lui, dans sa forme simple par des paramètres de contexte et dans sa forme structurée par des messages de contexte. Les paramètres de contexte représentent des concepts appartenant à une ontologie globale partagée par les services de l'environnement, tandis que les messages sont des structures de données qui encapsulent des paramètres de contexte. Les messages sont produits et consommés par les différents services concrets. Enfin, les utilisateurs sont aussi pris en compte par le modèle des connaissances via leurs spécifications que ce soit en termes de préférences de qualité de service ou bien en termes de règles événementielles. A travers

notamment cette possibilité d'établir des règles événementielles, le modèle de connaissances offre des points de jonctions entre d'une part, les objectifs (intentions) des utilisateurs lors de l'occurence d'éventuels événements dans l'environnement et d'autre part, les différentes configurations (composition) possibles des services disponibles.

La deuxième contribution a porté sur la proposition d'une architecture générale d'un système AmI que nous avons appelé *FrEvASeC* pour ***Fr**amework for **Ev**ents **A**ware **Se**rvice **C**omposition*. Le Framework *FrEvASeC* décrit ainsi notre vision du système de composition de services dans un environnement ambiant. Ce Framework est construit suivant un modèle multicouche afin de supporter un système adaptatif et capable d'effectuer le monitoring, la composition, la sélection et l'invocation des services appropriés lorsqu'un événement se produit dans un environnement ambiant. Plus précisément, *FrEvASeC* est structuré en cinq couches principales à savoir : la *couche physique*, la *couche connectivité*, la *couche gestion des ressources*, la *couche services*, et la *couche contrôle*. Cette dernière repose sur plusieurs modules interconnectés et engagés dans un processus de collaboration afin de réaliser les objectifs spécifiés lors de la détection des événements dans un environnement ambiant. Ainsi, *FrEvASeC* est un système qui réagit à la détection des événements et informe les utilisateurs concernés après l'exécution des services appropriés. Son principe général de fonctionnement repose sur une séparation claire entre trois catégories de services: les *services sensibles aux événements*, les *services sensibles au contexte* et les *services de contrôle des dispositifs*. Tous ces services sont contrôlés par le cœur du système *FrEvASeC* qui est constitué des modules de monitoring, de composition, de sélection et d'invocation des services. Par ailleurs, le système *FrEvASeC* est configurable par les utilisateurs et réactif aux événements. En effet, un utilisateur peut configurer certains paramètres du système et peut également recevoir des notifications en guise de retour du système en cas d'apparition d'un événement. Globalement, ce fonctionnement passe par quatre étapes principales : (i) spécifications des utilisateurs ; (ii) détection des événements ; (iii) gestion des événements détectés et (iv) notification des événements détectés.

La troisième contribution a porté sur la proposition d'un modèle de découverte et de classification de services permettant de localiser et de préparer sémantiquement les services nécessaires à la composition de services. Tout d'abord, la découverte de services se charge de la recherche, de la localisation et de la sauvegarde des services concrets. Son rôle principal consiste à mettre à jour dynamiquement et régulièrement la base de services concrets par les services qui apparaissent et disparaissent dans l'environnement ambiant. La découverte de service repose sur une architecture utilisant plusieurs répertoires locaux de services. Ces derniers sont découverts, lors du démarrage du système, par des requêtes de découverte active. Par la suite, une souscription (abonnement) est effectuée auprès des répertoires de services découverts pour écouter passivement les annonces spontanées relatives aux événements de publication, de modification ou de suppression de services. La classification de services se charge, quant à elle, de classifier les services déjà découverts dans des classes de services appelées services abstraits. Ainsi, la classification de services est une phase complémentaire qui suit la phase de découverte de services, et ce, afin de préparer les services abstraits nécessaires à la composition de services. Cette préparation des services se base sur un modèle de *matching* sémantique qui permet, d'abord, d'établir une similarité sémantique

entre les messages fournis et requis par les services découverts, puis, d'identifier les services qui sont fonctionnellement équivalents. Le modèle de *matching* repose essentiellement sur l'utilisation d'une ontologie globale des concepts. Cette ontologie permet de comprendre la signification sémantique des différents concepts véhiculés par les messages d'entrés/sorties des services découverts.

La quatrième contribution a porté sur le modèle de composition de services lui-même. Dans ce modèle, une composition de services est définie par un graphe où chaque service abstrait est explicitement lié à ses prédécesseurs par ses entrées et à ses successeurs par ses sorties. La construction d'un tel graphe s'effectue de façon automatique et dynamique en deux phases principales: la phase offline et la phase online. Dans la phase offline, un graphe global reliant tous les services abstraits disponibles est généré automatiquement en se basant sur des règles de décision sur les entrées et sorties des services. Ces règles visent l'optimisation à la fois du nombre de services et de messages qui apparaissent dans le graphe global. De plus, l'arrivée et la disparition des services sont gérées dynamiquement par des mises à jour partielles du graphe global. Dans la phase online, des sous graphes sont extraits spontanément à partir du graphe global selon les tâches à réaliser. Ces tâches sont indiquées au système sous forme de règles événementielles qui sont déclenchées par des événements qui surviennent dans l'environnement ambiant. Grâce à l'utilisation des règles événementielles par le mécanisme d'extraction des sous graphes guidé par des objectifs précis, il n'est pas nécessaire d'indiquer, a priori et d'une façon statique, au système les services à composer. C'est le système qui se charge d'identifier les services appropriés pour atteindre les objectifs spécifiés. Par conséquent, cette approche de composition de service présente plusieurs avantages. D'abord, le graphe global est construit en offline avant que le système ne reçoive des tâches à réaliser. Cela permet d'augmenter significativement les performances du système notamment son temps de réponse à une tâche donnée. En effet, il est plus rapide d'extraire un sous graphe pour réaliser une tâche en online dès lors que le graphe global est déjà en place. Ensuite, un sous graphe contient un ensemble de services abstraits qui peuvent avoir plusieurs réalisations concrètes. Ainsi, l'exécution d'un sous graphe est adaptée dynamiquement en fonction des services concrets disponibles au moment de cette exécution. Enfin, le graphe global assure la réalisation des tâches du point de vue fonctionnel, sans tenir compte, a priori, des aspects non-fonctionnels relatifs à la qualité de service et au contexte d'utilisation. Puisque ces derniers sont sujets à des variations continuelles, il est plus judicieux d'en tenir compte à des stades ultérieurs de la composition notamment au moment de la réception d'une tâche et au moment de son exécution.

La cinquième contribution a porté sur la proposition d'un modèle de sélection de services. Ce modèle constitue une étape complémentaire et nécessaire à la composition de services. En effet, la sélection de services permet l'exécution concrète des graphes de composition de services tout en garantissant une meilleure qualité de service. Le modèle de sélection de services proposé comprend trois phases principales : (i) estimation de la qualité de service ; (ii) sélection de services avec apprentissage, et (iii) invocation des services. La phase d'estimation de la qualité de service consiste à évaluer la qualité globale d'un service concret en tenant compte de ses paramètres statiques et dynamiques de qualité de service. La phase de sélection de services avec apprentissage consiste, quant à elle, à choisir le service concret à

invoquer. Il s'agit, en fait, de déterminer le meilleur service concret en termes de qualité de service en se basant sur l'estimation fournie par la phase précédente. De plus, la sélection de services utilise un mécanisme d'apprentissage Bayésien afin de tenir compte du caractère incertain des réponses des services concrets. Enfin, la phase d'invocation de services consiste à exécuter réellement le service concret sélectionné dans la phase prétendent en utilisant les valeurs des messages d'entrées de ce service. A la fin de cette exécution, les valeurs des messages de sortie du service exécuté sont reçues et sauvegardées pour une utilisation ultérieure. De plus, les valeurs des paramètres dynamiques de la qualité de ce même service sont mises à jour à partir de l'observation des différents paramètres de l'exécution du service.

La sixième et dernière contribution a porté sur la proposition d'un modèle de monitoring de services. Ce modèle permet de contrôler la composition, la sélection et l'invocation de services afin de garantir une continuité de service face aux pannes imprévisibles qui peuvent survenir dans un environnement ambiant. Le monitoring de services est réalisé suivant une stratégie flexible et tolérante aux pannes avec prise en compte du contexte courant d'utilisation. L'objectif de cette stratégie consiste à augmenter les chances d'exécution d'un graphe de composition de services dans un environnement dynamique et incertain tout en lui garantissant une meilleure qualité de service. Afin d'atteindre cet objectif, la stratégie de monitoring repose sur deux mécanismes de substitution de services : un mécanisme de substitution local et un mécanisme de substitution global. Le mécanisme de substitution local consiste soit à remplacer un service concret par un autre service concret dans le même service abstrait, ou bien à remplacer un service abstrait par un autre service abstrait fournissant certaines fonctionnalités similaires. Le mécanisme de substitution global consiste, quant à lui, à effectuer une recomposition de tous les services disponibles. Ces différents mécanismes de substitution sont intégrés dans une approche en couches où on distingue trois niveaux principaux de monitoring de services, à savoir: le niveau initial de monitoring de services, le niveau supérieur de monitoring de services et le niveau inférieur de monitoring de services. A chaque niveau, un ou plusieurs algorithmes sont intégrés pour effectuer une tâche spécifique. Ces algorithmes sont essentiellement fournis par les modèles de composition et de sélection de services. Ainsi, la stratégie de monitoring de services définit une coordination appropriée entre les modèles de composition et de sélection de services afin d'atteindre concrètement l'objectif désiré lorsque'un événement se produit dans un environnement ambiant.

La maison intelligente est choisie pour l'expérimentation et la validation des différentes contributions apportées par cette thèse car elle représente un environnement ambiant par excellence. Tout d'abord, la maison est un système ouvert et distribué : des utilisateurs et des équipements variés peuvent se trouver à différents endroits, être mobiles et y séjourner plus ou moins longtemps. Ensuite, ces équipements distribués sont hétérogènes : leurs capacités, leurs langages de programmation, leurs protocoles de communication et leurs sémantiques d'utilisation sont a priori hétérogènes. Enfin, la maison forme un système dynamique et incertain : les utilisateurs sont mobiles et les divers équipements qui s'y trouvent peuvent se connecter et se déconnecter de façon imprévisible au gré de leurs déplacements dans la maison ou en raison de pannes ou de mesures de sauvegarde d'énergie. Par ailleurs, la maison intelligente trouve tout son intérêt dans le domaine de l'assistance et du maintien de personnes à domicile. En effet, les utilisateurs vaquent à de nombreuses activités et peuvent être

assistées par divers équipements et services disponibles. L'intérêt est d'autant plus grandissant pour le maintien à domicile des personnes âgées à forte dépendance ; ces personnes nécessitant une surveillance permanente et particulière de l'évolution de leurs états de santé. Cette surveillance permet d'anticiper et de réduire les risques d'accidents liés au vieillissement et d'effectuer une prévention médicale suite à une évolution de l'état de fragilité de la personne âgée.

1.6. Travaux de recherche liés à ces contributions

1.6.1. Communications et Publications

- A. Yachir, Y. Amirat, A. Chibani, N. Badache, "Towards an Event-Aware Approach for Ubiquitous Computing based on Automatic Service Composition and Selection", Annals of Telecommunications, Springer, vol. 67, no. 7-8, pp. 341-353, 2012.
- A. Yachir, K. Tari, Y. Amirat, A. Chibani, N. Badache, "MDP and Learning Based Approach for Ubiquitous Services Composition", in Towards SmArt COmmunications and Network technologies applied on Autonomous Systems (SaCoNAS), workshop at IEEE GLOBECOM 2010, Miami, Florida, USA, December 6 – 10, 2010.
- K. Tari, Y. Amirat, A. Chibani, A. Yachir, A. Mellouk, "Context-aware Dynamic Service Composition in Ubiquitous Environment", in Proc. Of the IEEE International Conference on Communications (ICC), Cape Town, South Africa, May 23–27, 2010, pp. 1-5.
- A. Yachir, Y. Amirat, K. Tari, A. Chibani, "QoS Based Framework for Ubiquitous Robotic Services Composition", in Proc. Of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2009, St. Louis, USA, Oct. 11-15, 2009, pp. 2019 – 2026.
- A. Yachir, Y. Amirat, A. Chibani, K. Tari, "Approche basée sur la qualité de service pour la composition automatique des services en robotique ubiquitaire", in 7èmes Journées Nationales de la Recherche en Robotique, JNRR'09, Neuvy-sur-Barangeon, France, 4-6 novembre 2009.
- K. Tari, Y. Amirat, A. Chibani, A. Yachir, "Rule-based Approach for Automatic Service Composition in Ubiquitous Environment", in Proc. Of the 6th International Conference on Ubiquitous Robots and Ambient Intelligence, URAI 2009, Gwang Ju, Korea, October 29-30, 2009.
- A. Yachir, K. Tari, A. Chibani, Y. Amirat, "Toward an Automatic Approach for Ubiquitous Robotic Services Composition", in Proc. Of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2008, Nice, France, Sept, 22-26, 2008, pp. 3717-3724.

1.6.2. Exposés dans le cadre de conférences, séminaires, journées thématiques

- Présentation à la conférence : IEEE International Conference on Intelligent Robots and Systems (IROS 2008) du 22 au 26 Septembre 2008 à Sofia Antipolis, Nice, France.
- Présentation aux journées du Groupe de Recherche (GDR) sur la robotique ubiquitaire le 11 Juin 2009 à l'Université Pierre et Marie Curie (Paris VI).

- Présentation au Séminaire du laboratoire images, signaux et systèmes intelligents (LISSI) le 14 Octobre 2009 à Sénart (Paris).
- Présentation au 7ème Journées Nationales de la Recherche en Robotique (JNRR'09) du 04 au 06 Novembre 2009 à Neuvy-sur-Barangeon, France.
- Présentation au 5ème Journée du GT ARC (Automatique et Réseaux de Communication) le 22 Mai 2012 à l'ENSAM– Paris.

1.6.3. Encadrement des projets de fin d'études

- "Conception et réalisation d'une infrastructure de découverte et de composition dynamique de services dans un environnement intelligent.", *Mémoire de Projet de Fin d'Etude d'Ingéniorat EMP - Bordj el Bahri, Alger, Algérie,* 2009.
- "Localisation hybride des nœuds statiques et mobiles dans un réseau de capteurs sans fil", *Mémoire de Projet de Fin d'Etude d'Ingéniorat EMP - Bordj el Bahri, Alger, Algerie*, 2010.
- "Conception et réalisation d'une plateforme de services à base de la technologie DPWS (*Devices Profile for Web Services*)", *Mémoire de Projet de Fin d'Etude d'Ingéniorat EMP - Bordj el Bahri, Alger, Algérie*, 2011.
- "Approche MDA pour la modélisation et l'implémentation des processus métiers collaboratifs. Application : le processus d'approvisionnement électronique inter-entreprise (B2B)", *Mémoire de Projet de Fin d'Etude d'Ingéniorat EMP - Bordj el Bahri, Alger, Algérie*, 2011.
- "Sélection de services dans un environnement ubiquitaire.", *Stage de Master Recherche, Laboratoire LISSI, France, 2012.*
- "Conception et réalisation d'un compositeur de services dans un environnement ubiquitaire.", *Mémoire de Projet de Fin d'Etude d'Ingéniorat EMP - Bordj el Bahri, Alger, Algérie*, 2013.
- Membre de l'équipe du Projet National de Recherche (PNR) intitulé « PTEA: Plateforme de Test d'un Environnement Ambiant **»,** *PNR entre le Laboratoire IA-UERI / EMP et ANDRU/MESRS, Algérie 2012-2013.*

1.7. Organisation du manuscrit

Le mémoire de thèse est composé de huit chapitres. Le présent chapitre constitue une introduction générale qui précise le contexte de l'étude, les contributions de la thèse et l'organisation du mémoire. Le reste du mémoire est divisé en deux grandes parties : l'état de l'art et les contributions relatives à notre proposition.

Etant donné que les travaux de cette thèse se situent à l'intersection de plusieurs domaines, notamment les approches orientées services et les approches de composition de services dans les environnements AmI, la première partie qui présente l'état de l'art est structurée en trois chapitres :

Dans le deuxième chapitre, une analyse de l'évolution des middlewares utilisés pour le développement des applications dans le domaine de l'intelligence ambiante est présentée. L'accent est mis notamment sur les architectures SOA et leurs avantages pour la construction

d'applications en environnements ambiants. Dans ce cadre, nous présentons trois modèles qui permettent la mise en œuvre de SOA dans le cadre de l'intelligence ambiante : les services Web, le Web sémantique et les services Web pour dispositifs. A la fin de ce chapitre, les différents défis soulevés par l'intelligence ambiante sont traduits en modèles orientés service. La composition de services est située au cœur de ces modèles et constitue le noyau d'un système intelligent en orientation services.

Dans le troisième chapitre, nous présentons tout d'abord quelques définitions et concepts liés à la composition de services puis nous proposons une classification des différentes catégories de composition de services rencontrées dans la littérature. Nous étudions également les différentes approches de composition de services et un ensemble de travaux proposés pour chaque approche. Enfin, nous détaillons les mécanismes d'adaptation des compositions de services au contexte et justifions le choix des approches intergicielles pour la composition de services dans un environnement ambiant.

Dans le quatrième chapitre, nous passons en revue les différentes approches intergicielles pour la composition de services. Ces approches sont analysées suivant certains aspects en relation directe avec la composition de services notamment la sélection et le monitoring de services. Ensuite, nous présentons une analyse comparative des approches étudiées en termes de prise en compte de certains défis et exigences des environnements ambiants. Enfin, ce chapitre se termine par une conclusion générale synthétisant les travaux de l'état de l'art.

La deuxième partie porte, quant à elle, sur notre contribution, un Framework pour la composition de services dans un environnement AmI. La présentation de notre proposition est aussi divisée en trois chapitres :

Dans le cinquième chapitre, nous décrivons le modèle des connaissances et l'architecture du système de composition de services. Ainsi, l'organisation de ce chapitre est scindée en deux parties. La première définit et formalise les différents concepts adoptés par le modèle des connaissances. La deuxième partie décrit, quant à elle, sa vision du système de composition de services dans un environnement ambiant. Cette vision repose sur un Framework comprenant plusieurs modules interconnectés et engagés dans un processus de collaboration afin de réaliser les objectifs spécifiés lors de la détection des événements dans un environnement ambiant. Ainsi, le fonctionnement de ce Framework est détaillé à travers la description des relations liant les modules composant le Framework.

Dans le sixième chapitre, nous détaillons la stratégie globale proposée pour la composition de services dans un environnement ambiant. Ce chapitre décrit les modèles orientés services sur lesquels repose le fonctionnement du Framework de composition de services. Ces modèles, principalement au nombre de cinq, sont les suivants : le modèle de découverte de services, le modèle de classification de services, le modèle de composition de services, le modèle de sélection de services et enfin le modèle de monitoring de services. Ainsi, l'organisation de ce chapitre est scindée en quatre parties. La première partie porte sur le modèle de découverte et de classification de services. Cette partie décrit d'abord l'approche de découverte de services, puis celle de classification de services. La deuxième partie porte sur la formalisation du modèle de composition de services. Il s'agit d'une description technique de l'approche de composition de services par plusieurs algorithmes. La troisième partie porte, quant à elle, sur

le modèle de sélection de services qui se base sur trois phases : phase d'estimation de la qualité de service, phase d'invocation de services, et phase de sélection de services avec apprentissage. La quatrième et dernière partie porte sur le modèle de monitoring de services. Ce modèle comprend plusieurs phases de monitoring de services afin de garantir une continuité de service malgré les pannes imprévisibles qui peuvent survenir dans l'environnement.

Le septième chapitre est dédié à la mise en œuvre et à la validation expérimentale des modèles de composition, de sélection et de monitoring des services proposés. L'objectif principal est de montrer l'intérêt et la faisabilité du Framework proposé dans un environnement ambiant réel. Ce chapitre est organisé en trois parties : La première porte sur la description de la plateforme ubiquitaire développée au laboratoire LISSI. Cette partie détaille les différents dispositifs constituants la plateforme matérielle ainsi que les différentes parties de la plateforme logicielle, en particulier, l'ontologie des connaissances manipulées, les services abstraits et concrets ainsi que les messages échangés entre ces services. La deuxième partie porte, quant à elle, sur l'implémentation de plusieurs scenarii d'assistance d'une personne à domicile. Ces scénarii correspondent à des configurations différentes de services. Ces configurations sont générées automatiquement suite à l'observation de certains événements dans l'environnement ambiant. Chaque configuration de services est représentée par un sous-graphe qui est extrait automatiquement du graphe global représentant tous les services disponibles. L'exécution et le monitoring de certains scénarii sont également présentés. Enfin, dans la dernière partie, nous montrons, à travers une série de test d'évaluation des performances, l'apport et l'intérêt des différents algorithmes qui constituent le noyau de son Framework.

Enfin, dans le huitième et dernier chapitre, un bilan global des différentes contributions est dressé avec des perspectives de recherche découlant de ces travaux de thèse.

PARTIE I : ETAT DE L'ART

CHAPITRE 2

Intelligence ambiante et orientation services

2.1. Introduction

Le développement rapide de l'informatique a permis de passer de l'utilisation d'imposantes stations de travail à des ordinateurs de poche quasi invisibles. Aujourd'hui l'informatique et les équipements électroniques envahissent progressivement l'univers quotidien. Ces équipements peuvent intelligemment réagir à l'activité de l'utilisateur afin de l'assister dans ces activités de tous les jours. Ces progrès actuels des systèmes informatiques nous conduisent vers l'ère de l'informatique ubiquitaire et de l'intelligence ambiante, dans laquelle tous les équipements de l'environnement seront dotés d'une capacité de calcul et de communication. Cependant, ces équipements intelligents sont très variés et de nature distribuée et hétérogène. De plus, ils évoluent dans des environnements très ouverts, hautement dynamiques et incertains. Ainsi, la mise en place du concept de l'intelligence ambiante soulève des défis scientifiques considérables.

Certaines réponses à ces défis sont aujourd'hui offertes par de nouvelles approches intergicielles qui permettent de masquer notamment l'hétérogénéité et la distribution des différentes entités de l'environnement. Parmi ces approches, le paradigme orienté services constitue l'une des réponses prometteuses aux défis des environnements AmI. Ce paradigme est réalisé dans le cadre des architectures orientées services (SOA : *Service Oriented Architecture*). SOA est une architecture type qui offre des solutions au problème de l'interopérabilité entre les applications distribuées. Dans SOA, un service fournit un ensemble de fonctionnalités définies par une description publiée par un fournisseur de services. À partir de cette description, un consommateur de services peut alors rechercher le service qui répond à ses besoins, le sélectionner puis l'invoquer. Les services Web sont l'une des réalisations de l'architecture SOA. Ils représentent des services faiblement couplés qui communiquent en utilisant les technologies standards qui ont fait le succès du Web. De plus, ces services sont décrit dans le cadre du Web sémantique qui fournie des langages de représentation de la connaissance d'une façon claire et compréhensible. Enfin, une nouvelle génération des services Web appelée services Web pour dispositifs est apparue afin de représenter les capacités de chaque dispositif comme des services. Ainsi, la communication entre les dispositifs ambiants revient à échanger des services entre eux.

Dans ce chapitre, nous analysons l'évolution des approches intergicielles utilisées pour le développement des applications dans le domaine de l'intelligence ambiante. Nous mettons l'accent notamment sur les architectures SOA et leurs avantages pour la construction de ce genre d'applications. Dans ce cadre, nous passons en revue trois modèles qui permettent la mise en œuvre de SOA dans le cadre de l'intelligence ambiante : les services Web, le Web sémantique et les services Web pour dispositifs. A la fin du chapitre, nous présentons la traduction des défis soulevés par l'intelligence ambiante à des modèles orientés service.

2.2. Les intergicielles au secours de l'intelligence ambiante

2.2.1. Définitions et objectifs d'une couche intergicielle

La communication dans les environnements ambiants nécessite de faire fonctionner tous les dispositifs qui y sont disséminés. Pour rappel, ces environnements sont composés des équipements hétérogènes, formant un système distribué à grande échelle, ouvert et dynamique, dans lequel les ressources disponibles et le contexte d'utilisation changent fréquemment. Afin de simplifier l'écriture et la maintenance des applications pour les environnements ambiants, il est naturel de séparer ces application des couches de communication avec le matériel et le réseau au travers de couches logicielles intermédiaires appelées intergicielles. Le terme "intergiciel", ou middleware représente les couches logicielles du "milieu" qui s'insèrent entre les couches matérielles, dites hardware, et l'application logicielle finale (**Figure 2.1**). Ces couches fournissent un haut niveau d'abstraction permettant de masquer l'hétérogénéité des réseaux de communication, des ressources matérielles, des systèmes d'exploitation et des langages de programmation.

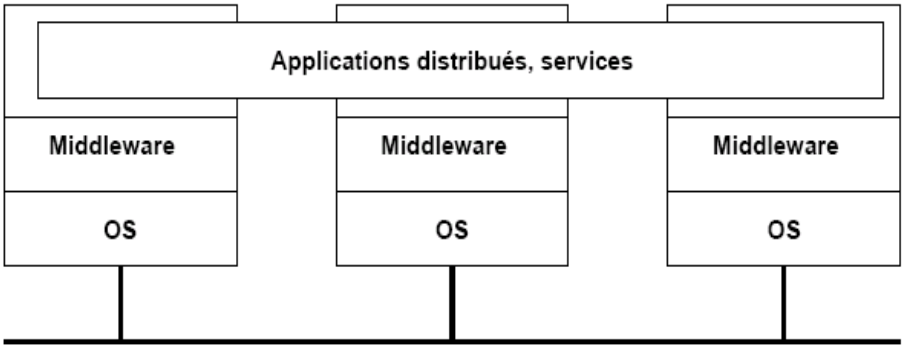


Figure 2.1 : Middlewares de communication

L'objectif principal des intergiciels consiste donc à masquer les aspects complexes de l'imbrication logicielle nécessaire aux applications complexes et fournir un support de communication interopérable permettant de simplifier l'intégration et la perception abstraite des entités de l'environnement. Pour ce faire, un intergiciel fourni un ensemble d'interfaces génériques masquant des implémentations correspondant à des environnements variés. Le développeur d'applications est simplement tenu d'implémenter certaines interfaces définies par l'intergiciel et de déclarer les besoins de l'application dans des fichiers de configuration au déploiement. Un intergiciel fourni également des fonctions complémentaires pour gérer les préoccupations non-fonctionnelles des applications telles que : la gestion des transactions, la sécurité, etc. Les Web Services sont un exemple d'intergiciel qui masque l'hétérogénéité des langages de programmation et la variabilité des implémentations **[45]**.

2.2.2. Intergiciels distribués

2.2.2.1. Intergiciels à objets répartis

Les middlewares à objets répartis se basent sur le principe d'appel de procédures à distance afin de pouvoir invoquer des objets distants de façon transparente. Ces objets présentent des interfaces qui masquent leurs détails d'implémentation et facilitent leurs

invocations. Ces middlewares sont devenus des standards. Trois d'entre eux se distinguent par le succès qu'ils ont eu : CORBA, RMI et DCOM.

CORBA : CORBA (*Common Object Request Broker Architecture*) est une architecture à objets distribués spécifiée par le groupe OMG (*Object Management Groupe*) **[46]**. C'est une norme de communication qui est utilisée pour l'échange entre objets logiciels hétérogènes. Un langage IDL *(Interface Definition Language)* décrit les traitements effectués et les formats de données en entrée et en sortie. Un bus applicatif, ORB *(Object Request Broker)* constitue le cœur de CORBA par lequel les requêtes sur les objets transitent. La fonction première de l'ORB est de répondre aux requêtes en provenance d'une application ou d'un autre ORB. Il constitue donc en quelque sorte un bus par lequel transitent les objets distribués entre clients et serveurs. Cette communication reste cependant totalement transparente pour le développeur CORBA **[47]**. En effet, une fois l'ORB initialisé, les objets distribués s'utilisent comme s'ils étaient locaux, quel que soit le langage dans lequel ils ont été implémentés. Assurer l'interopérabilité entre objets implantés sur des plates-formes hétérogènes nécessite de disposer d'un mécanisme permettant de décrire l'interface de ces objets indépendamment de leur implémentation. C'est là le rôle du langage IDL. Lorsqu'un client invoque un objet distant, une implémentation de l'interface de l'objet, appelé souche (*stubs*) permet d'empaqueter les paramètres de la méthode invoquée et d'appeler l'objet distant en passant en premier par les squelettes (*skeletons*) qui font l'opération inverse (dépaquetage des données), et transmet l'appel à l'interface de l'objet. Au retour, les données sont empaquetées par les squelettes et transmis au client par l'intermédiaire de la souche qui dépaquette les résultats fournis.

RMI : Le modèle de calcul de Java c'est « écrire une fois, exécuter partout ». Ici, le modèle d'objets distribués est intégré dans le langage lui-même, en essayant de garder le plus possible la sémantique des objets locaux. Le modèle Java étant incapable de spécifier comment initier des calculs dans un espace d'adressage distant, Sun a définit un mécanisme appelé RMI **[48]** *(Remote Method Invocation),* qui permet au programmeur d'appeler des objets distants et le renvoi du résultat d'une méthode exécutée dans une machine virtuelle différente de celle de l'objet l'appelant. RMI permet de créer des applications Java distribuées. Ces dernières implémentent les méthodes des objets Java distants qui peuvent êtres invoqués à partir de n'importe quelle machine qui possède la JVM (*Java Virtual Machine*).Un programme Java peut faire un appel à un objet distant une fois qu'il a obtenu sa référence, soit en cherchant dans un annuaire de services fourni par RMI, ou bien en recevant la référence comme argument en retour d'un appel. Les souches et squelettes, qui représentent respectivement les cotés client et serveur sont des classes compilées par le compilateur RMI (RMIC). Un client peut alors invoquer un objet distant qui se trouve dans un serveur. Le serveur lui même peut devenir client pour d'autres objets distants. RMI utilise la sérialisation d'objets pour empaqueter et dépaqueter les paramètres. Le protocole utilisé pour la communication est JRMP (*Java Remote Methode Protocol).* La programmation avec RMI est facile une fois le développeur a acquis une certaine expérience avec le langage de programmation Java et les applications distribuées. RMI ne contient pas un langage abstrait tel qu'IDL pour décrire les objets distants. De plus, il supporte un nettoyage distribué des objets en mémoire. Cependant, RMI peut être utilisé uniquement avec le langage Java de part

et d'autre de la connexion, et sa facilité d'utilisation provient du fait que ce middleware est maintenu simple et ne fournie pas des services comme CORBA. Cet aspect doit être pris en compte par le développeur.

DCOM : *Distributed COM* (DCOM) **[49]** est une version distribuée du modèle de Microsoft COM (*Component Object Model)* **[50]**, qui à été crée pour permettre la communication entre les applications Windows dans un environnement à base de composants. DCOM a étendu cette communication interprocessus à travers le réseau. DCOM permet d'appeler des objets distants en utilisant une couche de haut niveau au dessus du mécanisme DCE RPC *(Remote Procedure Call)* qui interagie avec les services de COM. Un serveur DCOM publie ses méthodes pour les clients sous forme d'interfaces. Ces dernières sont écrites en IDL, qui est similaire au langage C++. Le compilateur IDL crée des souches et des squelettes comme le compilateur IDL de CORBA. Mais la particularité de DCOM c'est que toutes les configurations de ses applications sont enregistrées dans la base de registre du système. L'utilisation de cette base de registre et le fait que DCOM soit une extension de COM, rendent exploitable les applications COM dans un environnement réparti. De plus, des librairies sont crées sous forme de fichiers qui décrivent l'objet distant qui peut être invoqué par le mécanisme COM. Un autre élément essentiel dans l'architecture DCOM est le Service Control Manager qui permet de gérer les composants du système (localisation, activation, enregistrement de référence). Le protocole utilisé est ORPC *(Object Remote Procedure Call).* Le niveau de spécification binaire de DCOM permet l'utilisation de plusieurs langages pour coder les objets du serveur. Les interfaces binaires de DCOM peuvent être considérées comme des tables de pointeurs vers les implémentations des méthodes de l'interface. Comme RMI, DCOM supporte le nettoyage distribué des objets distants en mémoire. Cela est réalisé par un mécanisme qui vérifie si le client est encore actif. Du coté serveur, un conteur de référence est maintenu pour les clients. S'il atteint zéro, l'objet sera détruis. La plupart des utilisateurs associent DCOM au système d'exploitation Windows. Toutefois, il existe des ports pour les plateformes Unix, VMS, Apple, et Macintosh.

2.2.2.2. Intergiciels à composants

La notion de composant étend la notion d'objet et se place à un niveau plus haut de modularité. Cette notion est apparue après le constat que l'approche objet a une granularité très fine, qui la rend très peu adaptée aux systèmes complexes et aux systèmes distribués en particulier. Ces derniers sont composés de plusieurs entités coopérants entre elles, et une fine granularité rendrait la conception et la maintenabilité difficiles, surtout si le développement de l'application est réalisé par plusieurs parties. Ainsi, la programmation orientée composants répond au problème de la complexité de gestion des codes métiers des applications. Elle a pour objectif d'une part, de séparer les aspects fonctionnels de l'application (le code métier) des aspects non fonctionnels (sécurité, transaction, cycle de vie, etc.) et d'autre part, de permettre la réutilisation des objets métiers dans de nouvelles applications. Un composant est un module logiciel autonome qui assure un service dans le système. Il est à la fois facile à créer et à maintenir et peut être déployé sur différentes plates-formes, en vue de son utilisation ou de sa réutilisation **[51, 52]**. Un composant offre une structure de programmation souple et unifiée, qui exporte les différents attributs ou méthodes sous forme d'interfaces, lui permettant d'interagir avec d'autres composants. Le composant encapsule un état et exhibe un

comportement à travers ses interfaces. Dans l'architecture d'un composant, on distingue deux parties distinctes : la première concerne le code métier du composant et la deuxième, le code non fonctionnel du composant. Cette deuxième partie est gérée par un conteneur qui a pour rôle de gérer le cycle de vie des composants et des connexions inter-composants.

Il existe principalement deux modèles d'intergiciels à composants : le modèle à composants métiers et le modèle à composants génériques appelés aussi hiérarchiques. Le modèle à composants métiers appelés aussi composants « gros grains » a été proposé pour le développement des systèmes répartis complexes à large échelle par l'assemblage de composants. Comme modèles de composants métiers, on peut citer : CCM (CORBA Component Model) **[53, 54]**, EJB de Sun **[55]** ou Microsoft .NET **[56]**. Le modèle à composants génériques offre quant à lui plusieurs niveaux d'abstraction à travers différentes granularité de composants qui peuvent aller des composants « petit grain » (dotés de conteneurs sans services de gestion) pour la conception de plateformes intergicielles, jusqu'à des composants « gros grain » ou composites dotés de conteneurs avec plusieurs services de gestion pour la conception de systèmes répartis complexes. Comme modèles de composants génériques, on peut citer : Fractal **[57]** et ArcticBeans **[58]**. La notion de composants composites permet de construire des composants métier (gros grains) à partir de la composition (assemblage) de plusieurs composants.

2.2.2.3. Intergiciels à agents

Les systèmes multi-agents (SMA) est un paradigme émergeant de la recherche en intelligence artificielle distribuée dans les années quatre-vingt. La notion d'agent, sur laquelle se fondent les systèmes multi-agents, s'est vue donner des définitions par plusieurs auteurs **[59, 60]**. Ce qu'il faut en retenir c'est qu'un agent, contrairement à d'autres programmes, doit au moins simultanément être : (1) situé : il perçoit le monde/environnement dans lequel il se situe ; (2) autonome : il prend des décisions et agit sur son environnement en vue d'atteindre son objectif ; et (3) interactif : il a la capacité d'interagir avec d'autres agents.

Partageant un espace commun, usuellement appelé environnement, les agents sont amenés à coopérer pour s'aider mutuellement dans l'accomplissement de leurs objectifs, partager leurs ressources, résoudre les conflits éventuels ou les éviter, etc. Ceci amène les agents à se coordonner afin de pouvoir évoluer en complète synergie dans cet environnement **[61]**. La coordination peut également être définie comme étant la gestion des interdépendances entre les activités **[62]**. Actuellement, il existe diverses formes et mécanismes de coordination; de la coordination implicite, comme dans les communautés biologiques (par exemple, les fourmis), à la coordination explicite à travers des mécanismes de raisonnement formalisés et plus élaborés.

Le modèle d'intergiciel à agents a pour objectif d'offrir au développeur une abstraction du niveau fonctionnel du système, lui permettant de se focaliser essentiellement sur la mise en œuvre de fonctionnalités ou de comportements de haut niveau des applications. Outre la création et la définition des comportements des agents, ce type d'intergiciel permet de définir le modèle de coordination souhaité entre les différents agents et supporte les communications entre ces derniers. En effet, les integiciels à agents assurent l'interopérabilité entres des systèmes multi agents hétérogènes par le biais de langages de communication de haut niveau.

De nombreux intergiciels à agents ont été mis en œuvre à partir d'intergiciels à objets ou à composants (Actor Foundry, MadKit, Jade, Jack10, Zeus, IBM Aglet, FIPA-OS, MAST, etc.). La majorité de ces intergiciels recourent à l'utilisation de la communication explicite par envoi de messages dont la syntaxe et la sémantique sont régies par des standards d'interopérabilité et de communication multi agents tels que KQML**[63]**, FIPA-ACL **[64]**, etc.

2.2.3. Intergiciels orientés services

2.2.3.1. La notion de service

Un service est une brique logicielle autonome qui fournie une fonction bien définie **[65]** telle que l'analyse d'informations, la recherche d'informations, etc. Un service est aussi considéré comme une entité logicielle autonome dotée d'une interface bien définie qui peut être accessible sans aucune connaissance de sa technologie sous-jacente. Un service est autonome dans la mesure où il peut se suffire à lui même, toutefois il peut être publié et rendu disponible pour être utilisable par des tiers. Ainsi des services peuvent être utilisés tels quels ou bien être composés pour mener à terme un processus complexe et atteindre un objectif précis de plus haut niveau. D'un point de vue implémentation, un service peut être aussi défini comme étant un module qui peut être invoqué, qui est assigné à une fonction spécifique, et qui offre une interface bien définie. L'implémentation de la fonction fournie est libre, plusieurs services peuvent la réaliser en utilisant différents paradigmes : objet, composant ou agent. Un service est décrit et utilisé indépendamment de sa réalisation grâce aux mécanismes d'encapsulation et de liaison retardée. La liaison proprement dite entre le fournisseur et le demandeur de service n'est effectuée qu'au moment de l'exécution. Ainsi, les liaisons se font et se défont dynamiquement lors de l'exécution des services. Un service est décrit par son descripteur de service, sa spécification en quelque sorte. Le descripteur comporte des :

- *Informations fonctionnelles* : la sémantique des opérations, le comportement du service (pré condition, post-condition, invariant, exception, propriétés), l'interface du service.
- *Informations non fonctionnelles* : prix, politique, spécification des mesures de la qualité de service, information de déploiement, etc.
- *Informations additionnelles* : composées d'informations sur le service, non spécifiées par le fournisseur du service telles que les notes, les rapports d'utilisation, etc.

2.2.3.2. La notion de SOA

L'architecture orientée service SOA (*Service Oriented Architecture*) est la partie architecturale de l'approche orientée service. Cette approche peut être appliquée dans le processus de développement logiciel depuis l'analyse jusqu'à l'implémentation. Le terme "SOA" est souvent employé pour désigner l'approche orientée service dans son ensemble. L'approche SOA, également connue sous différents noms tels que la programmation orientée service : SOP (*Service Oriented Programming*) et l'informatique orientée service : SOC (*Service Oriented Computing*), vise comme premier objectif d'assurer l'interopérabilité entre systèmes hétérogènes. L'idée fondamentale derrière SOA est de subdiviser le système en parties distinctes représentant une décomposition fonctionnelle du système original. Cette décomposition dont les parties s'appellent services, vise à avoir une séparation propre des fonctionnalités qui devraient favoriser la réutilisation de ces services pour la construction ou

l'amélioration de nouvelles applications. Dans la littérature, plusieurs définitions existent pour l'architecture orientée services. Dans **[66]**, SOA est définie comme étant *« une architecture logicielle s'appuyant sur un ensemble de services simples. Son objectif est de décomposer une fonctionnalité en un ensemble de fonctions basiques (les services) fournies par des composants et de décrire le schéma d'interaction entre ces services ».*Voici une autre définition de SOA: « *L'architecture orientée service est essentiellement une collection de services. Ces services communiquent entre eux à l'aide de données simple ou d'autres services coordonnant une certaine activité. SOA n'est pas quelque chose de nouveau, la première architecture orientée service pour beaucoup de personnes dans le passé était DCOM ou CORBA*» **[65]**. Les deux définitions précédentes mettent l'accent sur l'aspect communication entre les services. Cette communication peut soit consister en un simple passage de données ou impliquer la coordination de deux ou plusieurs services pour l'accomplissement d'une activité. Nous constatons également que l'approche SOA prend ses origines dans les anciennes architectures. Beaucoup considèrent que la première SOA est apparue avec l'utilisation des DCOM ou ORB basés sur les spécifications de CORBA. L'approche SOA vient pour combler le manque de canevas architectural permettant de : guider la conception, développer, intégrer et réutiliser des applications plus facilement et plus rapidement. Ainsi, une architecture orientée service permet d'assembler des composants et des services pour construire dynamiquement des applications.

2.2.3.3. Principe de SOA

L'architecture SOA est structurée autour de trois acteurs fondamentaux décrits dans la **figure 2.2**: *Le fournisseur de services* (*Service Provider*), *le demandeur (consommateur ou client) de services* (*Service Client*) et *le registre (ou répertoire) de service (Service Registy). Le fournisseur de services* est l'entité logicielle qui mette à disposition des services, le *demandeur de service* est l'entité cliente qui cherche à consommer un service spécifique, et le *registre de service* est l'entité ayant des informations sur les services disponibles et la façon d'y accéder. Le paradigme d'orientation service définit un schéma d'interaction (**Figure 2.2**) entre fournisseur de service, demandeur de service et le répertoire de service qui partagent la connaissance d'une interface de service. Un fournisseur de service développe le service et publie son interface dans le registre de service. Le consommateur de service aussi appelé utilisateur accède à ce registre afin de chercher des services qui satisfassent la tâche qui veut accomplir. Ainsi, l'interaction entre le client et le fournisseur est réalisée de façon dynamique grâce à l'abstraction procurée par l'interface de service.

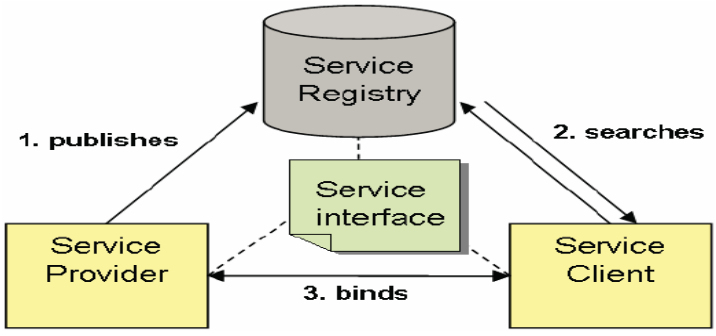


Figure 2.2 : Acteurs classiques d'une architecture orientée services

Un rôle supplémentaire introduite dans le cadre de l'extension de SOA **[67]** est identifié comme *service agrégateur* (*Service Aggregator*), qui est le rôle d'une entité qui compose des services existants et les offre comme des nouveaux services pour les applications clientes. Comme le montre la **figure 2.3**, le *service agrégateur* agit à la fois comme prestataire de services (*Provider*) en fournissant des services composites à des applications et en tant que demandeur de service (*Requester*) en consommant les services disponibles.

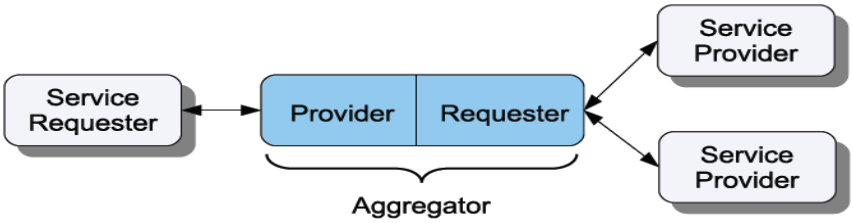


Figure 2.3 : Role du *Service Aggregator*

La **figure 2.4**, basée sur **[67]**, présente plus en détail la notion de service et illustre les fonctionnalités de base réalisées par les acteurs SOA. Pour permettre l'identification des services fournis et requis par les fournisseurs et les demandeurs, les services sont décrits de façon structurée en utilisant un langage ou un formalisme de description de services. Cette description contient les spécifications suivantes : la capacité du service, l'interface du service, le comportement du service, les propriétés de la qualité de service (QoS) ainsi que l'adresse pour accéder au service. La capacité du service décrit les fonctionnalités fournies par le service, c'est à dire, ce que le service fait. L'interface du service décrit la liste des opérations par lesquelles le service réalise sa capacité. Une opération d'un service représente l'unité de l'interaction avec le service, elle a une signature, soit une structure en termes de données à échanger avec le service. Le comportement du service, appelé aussi « conversation », définit les relations temporelles et les propriétés entre les opérations du service nécessaires à une interaction valable avec le service. Les propriétés de QoS du service décrivent les caractéristiques non fonctionnelles du service, tels que la sécurité ou les propriétés transactionnelles.

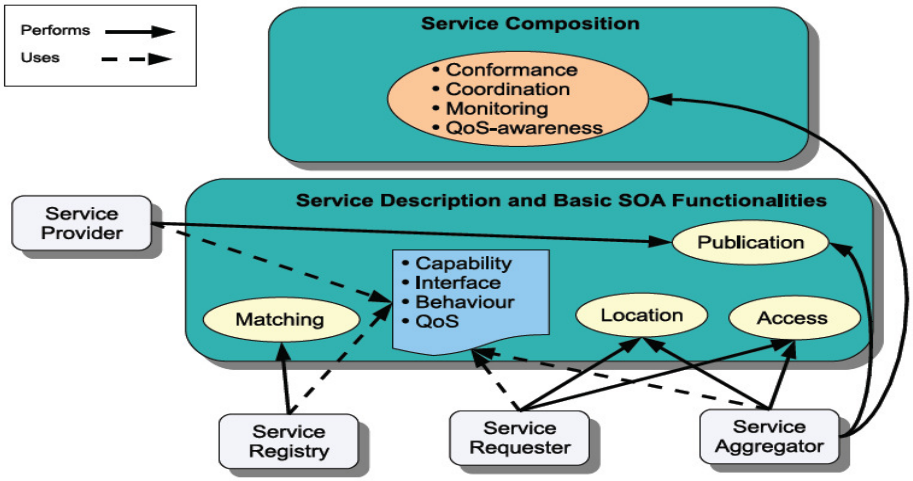


Figure 2.4 : Les éléments conceptuels de SOA

Dans la **figure 2.4**, les acteurs SOA sont associés aux fonctionnalités de base qu'ils effectuent. Ces fonctionnalités sont les suivantes:

1. *Publication du Service (Service Publication)*: permet aux fournisseurs de services d'enregistrer leurs services dans un registre de service.
2. *Localisation du Service (Service Location)*: permet aux demandeurs de services de récupérer les services souhaités à partir d'un registre de service.
3. *Matching du Service* (*Service Matching*): réalisé par un registre de service, permet de sélectionner parmi les services enregistrés, ceux qui correspondent mieux à une demande de service.
4. *Accès au Service* (*Service Access*): permet à un demandeur de service d'établir une connexion (i.e., *Service Binding*) avec le service sélectionné. Après l'établissement de cette connexion, l'interaction avec le service aura lieu comme un ensemble d'invocations successives d'opérations du service.
5. *Composition de service (Service Composition)*: permet l'intégration de plusieurs services en un seul service composite, qui peut être réalisé au moment de la conception (statique) ou lors de l'exécution (dynamique). La composition de service se décompose en quatre autres sous-fonctions:
   - *Conformité de Service* (*Service Conformance*) : garantit l'intégrité d'un service composite en évaluant la compatibilité de sa description avec celle de ses services composants.
   - *Coordination des services (Service Coordination)* : contrôle l'exécution des services qui prennent part à une composition de service.
   - *Surveillance de Service* (*Service Monitoring*) : permet d'observer l'état d'exécution des services composites pour déclencher éventuellement l'adaptation des services.
   - *Sensibilité à la QoS (QoS-awareness*) : vérifie le respect des exigences de QoS des services composites en fonction de la QoS fournie par leurs services composants.

2.2.3.4. Comparaison entre SOA et les intergiciels distribués

La principale différence entre SOA et les autres architectures distribuées, telles que CORBA, est le couplage lâche (faible) des services. Dans le développement des logiciels, le couplage se réfère au degré de dépendance entre les différents composants et modules du logiciel. Le paradigme d'orientation service se base sur des mécanismes de découplage des entités fournissant des fonctionnalités, appelées services, et les entités composant et consommant ces fonctionnalités. Le découplage repose sur la définition et la publication d'interfaces de services séparées des implémentations. Les interfaces de services décrivent un ensemble d'opérations et les qualités non-fonctionnelles. Les clients ou demandeurs de services requièrent les services selon des critères définis par l'interface de service – nom de l'interface, niveau de qualité de service, etc. Le répertoire de service répond aux requêtes de services des clients en retournant la liste des fournisseurs de service enregistrés qui correspondent aux critères du demandeur. Les liaisons proprement dites entre fournisseurs et demandeurs de services se font et se défont lors de l'exécution (liaison tardive). Ce schéma d'interaction rend la composition de service plus flexible **[45]**.

La vision d'un système destiné à traiter toute application comme un ensemble de services faiblement couplés a de nombreux avantages ; le plus évident est de pouvoir réutiliser un service pour plusieurs applications. Ainsi on peut concevoir les programmes comme des modules. Ils sont indépendants (leur état ne dépend pas des autres) et peuvent être distribués sur différentes machines. Les SOA sont souvent utilisées pour mettre en place des applications réparties. Ces applications sont construites à partir de services individuels exposant leurs fonctionnalités à travers des registres en faisant complètement abstraction de leur implémentation sous-jacente. Leurs interfaces publiées peuvent alors être recherchées et invoquées par des utilisateurs ou d'autres services. En utilisant plusieurs fournisseurs en redondance, on accroit la rapidité et la fiabilité d'un système. Il est possible également d'utiliser un logiciel de composition qui crée les liens entre fournisseurs et consommateurs ; les programmes ne sont alors plus statiques. Dans ce cas, plusieurs services peuvent être utilisés ensemble d'une manière coordonnée. Le service agrégé ou composé peut être utilisé pour satisfaire des exigences plus complexes de l'utilisateur. Il est possible de changer de fournisseur pour une fonction précise, éventuellement pendant l'exécution du programme. Outre le couplage lâche qui semble être la marque saillante des approches à service, d'autres différences avec les architectures distribuées sont illustrées sur la **table 2.1**.

Architecture distribuée	**Architecture orientée service**
Orientée fonctionnalité	Orientée processus
Elaborée pour durer	Elaborée pour gérer les changements
Processus de développement long	Processus de développement court
Centrée sur la réduction des couts	Centrée sur le support aux métiers
Organisée au sein d'une application	Permet la composition
Système fortement couplé	Système faiblement couplé, agile
Orientée objets	Orientée messages
L'implémentation doit être connue	L'implémentation reste abstraite
Technologie homogène	Technologie hétérogène

Table 2.1: Architecture SOA et Architecture distribuée.

Plusieurs technologies peuvent être employées afin de mettre en application une SOA. Les Web Services sont un exemple phare de ces technologies qui ont fait un large succès auprès des grands acteurs de l'informatique. Ainsi, grâce à cette technologie, les services vont pouvoir être implémentés sous la forme de services Web. Ces derniers masquent l'hétérogénéité des langages de programmation et la variabilité des implémentations et assurent l'interopérabilité entre systèmes hétérogènes. Cette interopérabilité est mise en œuvre à travers le protocole HTTP et le langage XML. Le protocole HTTP permet de gérer les messages (HTTP POST, HTTP GET, HTTP CONNECT,..) entre le consommateur et le fournisseur du service avec une garantie de sécurité et de traversé des environnements réseaux hostiles (NATs, Firewall, routeurs) ; on parle alors d'interopérabilité au niveau applicatif. Le langage XML permet, quant à lui, de définir des interfaces de services (ou des descriptions de services) et les structures de données associées, permettant à d'autres systèmes de découvrir et d'invoquer ces services ; on parle alors d'interopérabilité au niveau syntaxique. L'avènement,

ces dernières années, d'une nouvelle technologie appelée Services Web pour Dispositifs comme UPnP et DPWS a donné un second souffle aux services web quant à leur mise en œuvre sur des dispositifs physiques. En effet, grâce à ses mécanismes de découverte distribuée de services et de communication événementielle, cette nouvelle technologie permet d'inventorier et de contrôler dynamiquement les dispositifs présents dans un environnement. Les apports et le principe de ces deux technologies en l'occurrence les services web et les services web pour dispositifs sont énoncés dans les sections suivantes.

2.3. Les services Web

2.3.1. Définitions et caractéristiques

L'émergence des services Web a donné une grande importance à l'architecture SOA comme approche universelle de développement des applications distribuées **[68]**. Un service Web est un service qui communique avec des clients à travers un ensemble de protocoles et de technologies standards. Ces standards sont supportés et implémentés par les principaux acteurs du monde du logiciel pour l'entreprise comme Microsoft, IBM ou SUN. L'objectif ultime est de transformer le Web en une infrastructure universelle permettant de déployer des logiciels distribués (services) qui ont la capacité d'interagir de manière intelligente et efficace. Cette universalité a rendu ainsi les services Web comme l'approche dominante pour implémenter une SOA. Le terme service est de plus en plus associé aux services Web **[69]** qui sont actuellement largement utilisés et devenus un standard de fait. Selon IBM **[70]**, « *Web services sont auto contenus, applications modulaires, accessibles via le Web à travers des langages standards et ouverts, qui fournissent un ensemble de fonctionnalités pour les entreprises ou les individus* ». Cette définition met l'accent sur le faite que les services Web sont des applications ouvertes accessibles par d'autres applications via le Web. Une autre définition qui raffine plus la définition précédente est celle donnée par W3C (*World Wide Web Consortium*) **[71]** : « *un service Web est un système logiciel identifié par un URI, dont les interfaces publiques et leurs liaisons sont décrites en utilisant XML. Sa définition peut être découverte par les autres systèmes logiciels. Ces systèmes peuvent alors interagir avec le service Web avec la manière décrite dans sa définition, en utilisant des messages basés sur XML et transmis par des protocoles internet* ». La définition de W3C insiste sur le fait que les services Web doivent être capables d'être identifiés, décrits, et découverts.

Les services Web développés par différentes organisations peuvent être intégrés pour satisfaire les exigences des utilisateurs. Cette intégration est basée sur les standards communs des services Web, indépendamment des langages d'implémentation, et des plateformes d'exécution. En général, les services Web possèdent les caractéristiques suivantes qui leurs permettent une meilleure intégration dans des environnements hétérogènes :

- **Faiblement couplé** : Comparativement aux composants fortement couplés tels que DCOM ou CORBA, les services Web sont autonomes et peuvent opérer indépendamment les uns des autre. La caractéristique de faible couplage permet aux services Web de localiser d'autres services dynamiquement, et de communiquer avec eux.

- **Accessibilité universelle** : les services Web peuvent être identifiés, décrits, et découverts à travers le Web qui permet une accessibilité facile. La description et l'annonce (publication) des services Web permettent aux utilisateurs du Web de localiser les services appropriés.
- **Langages standards** : quoique le noyau des services Web puisse être implémenté par différents langages de programmation, la description de leurs interfaces se base sur un langage uniforme et standard qui est XML.

2.3.2. Langages standards des services Web

Les services Web respectent les principes de l'approche orientée service précédemment présentés. Ils sont donc décrits, publiés et découverts. Le fournisseur doit d'abord décrire le service Web à fournir à l'aide d'un langage de description appelé WSDL *(Web Services Definition Language)* qui se base sur une grammaire XML *(Extensible Markup Language)*. Il doit ensuite publier cette description du service dans un registre de service appelé UDDI *(Universal Discovery Description Integration)*, et ce, afin de rendre son service disponible et connu par les clients. Ces derniers peuvent alors accéder au registre UDDI et rechercher le service Web désiré. Après l'avoir trouvé, ils utilisent sa description pour créer un lien avec le fournisseur de ce service à l'aide d'un protocole de communication appelé SOAP *(Simple Object Access Protocol)* qui enveloppe les messages dans un format XML.

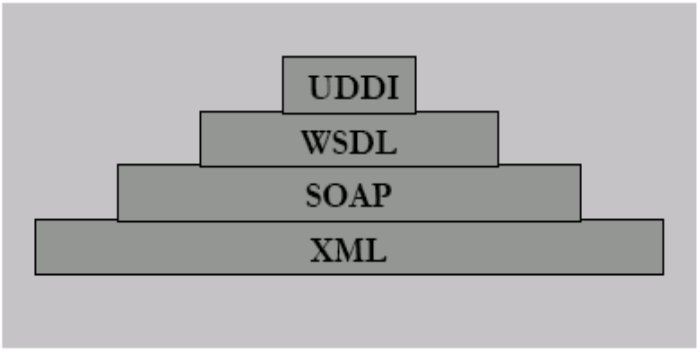


Figure 2.5 : Pile des standards des services Web

Les différentes couches qui constituent la pile des standards des services Web (**Figure 2.5)** sont : XML, SOAP, WSDL et UDDI.

XML : XML est une norme du W3C depuis 1998, constitue un métalangage pour définir des langages de balisage. Il conserve la syntaxe de SGML et reprend ses qualités à savoir : un balisage logique hiérarchique, une description facultative du document qui peut être extérieure ou intégrée au document, une portabilité quasi universelle. C'est ainsi que les fonctionnalités les plus lourdes et les plus rarement utilisées de SGML ont été supprimées dans XML. Ce dernier permet également de créer des pages Internet sophistiquées, de structurer un document, de décrire des données. Il est aussi utilisé pour les échanges entre machines et/ou programmes, mêmes étrangers entre eux. XML facilite l'utilisation des services Web : d'abord par la séparation du contenu d'un document, de sa structure et de sa représentation qui facilite l'échange d'informations entre les différents partenaires. Ensuite, en permettant la composition de services plus complexes à travers la définition des

enchaînements entre les services. Enfin, en favorisant l'émergence de standards d'industrie dans la mesure où les structures de données sont réutilisables.

SOAP : SOAP est un standard proposé par W3C pour l'échange des données. Ces dernières sont d'abord enveloppées dans des messages en format XML, puis échangées par des appels de procédures à distance (RPC) en utilisant HTTP/SMTP/POP comme protocole de communication. SOAP est la plupart du temps utilisé sur la couche de transport HTTP et il n'est pas perturbé par les pare-feux *(firewall).* Il est aussi indépendant de la plate-forme et du langage des services. Si un consommateur souhaite utiliser un service Web, il va construire avec une application cliente un message SOAP conforme au document WSDL du service. Un message SOAP correspond à une enveloppe contenant jusqu'à 2 parties : la partie *header* dans laquelle il est possible d'ajouter ou de modifier des informations contextuelles (signatures, destinataires…) et la partie *body* qui contient le corps du message à transmettre.

WSDL : WSDL est un langage de description des services proposé par W3C. Il se base sur une grammaire XML pour décrire de manière abstraite et indépendante du langage de programmation, l'ensemble des fonctionnalités offertes par un service. Il permet de connaître les protocoles, les serveurs, les ports, le format des messages, les entrées, les sorties, les exceptions possibles et les opérations réalisées par un service Web. Ainsi, WSDL propose une double description du service : une vue abstraite *(le quoi)* qui présente simplement les opérations et les messages des services et une vue concrète *(le comment, et le où)* qui présente les choix d'implémentation faits par le fournisseur du service (détermination du protocole et du mode d'accès).

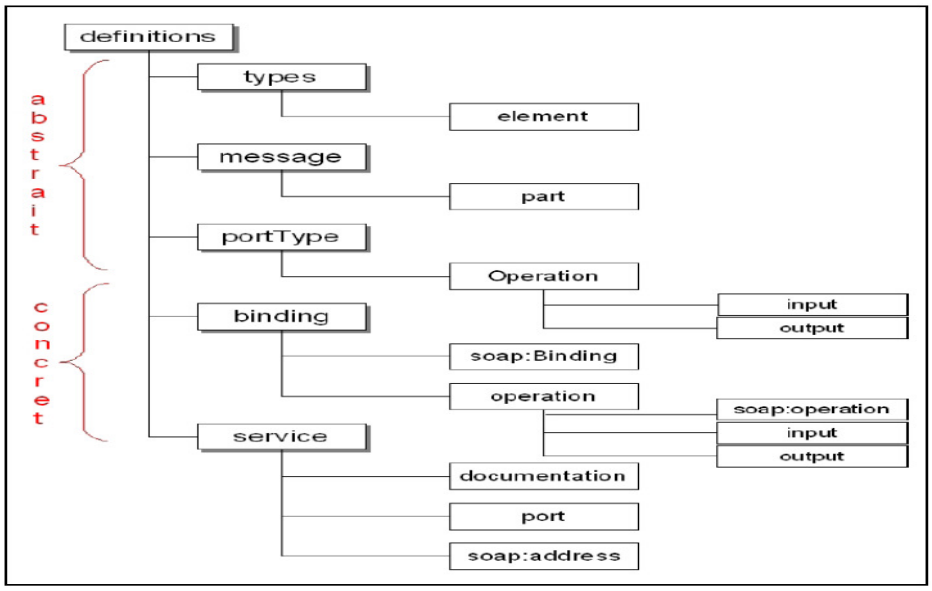


Figure 2.6 : Architecture plus détaillée d'un fichier WSDL.

La **figure 2.6** correspond à la structure d'un fichier WSDL qui comprend deux parties : la partie abstraite et la partie concrète. La partie abstraite décrit les types de données utilisés, les messages échangés et les opérations possibles pour un service Web. Cette partie est définie par les balises : *types*, *messages* et *port types*. Les *types* fournissent la définition de type de données utilisées pour décrire les messages échangés. Les *messages* représentent une définition abstraite (noms et types) des données à échanger. Les *ports types* décrivent un

ensemble d'opérations abstraites. Chaque opération possède au plus un message en entrée, et peut avoir plusieurs messages en sortie. La partie concrète, qui est spécifique à l'implantation, comprend les balises : *binding* et *service*. L'élément *binding* spécifie le protocole avec lequel les clients peuvent invoquer le service. Cette élément réalise une liaison entre un *port type* et a un protocole concret (SOAP, HTTP, MIME). L'élément *service* permet d'associer au service Web une adresse réseau.

UDDI : UDDI est un registre (annuaire) qui décrit comment publier et découvrir un service Web. Les fournisseurs de services peuvent alors enregistrer et publier leurs services dans cet annuaire. UDDI contient des métadonnées (noms, IDs, catégories, types, etc.) qui peuvent être utilisées pour la recherche, la sélection et l'invocation des services. Ces métadonnées sont structurées dans trois types de pages : blanches, jaunes et vertes. Chaque type de pages donne des informations de nature différentes. Les pages blanches contiennent des informations sur le fournisseur du service (nom, adresse, etc.), les pages jaunes indiquent ce que fait le service, et les pages vertes, plus techniques, décrivent comment utiliser un service Web et les moyens d'interaction avec un fournisseur en se basant sa description fournis en WSDL. La recherche se fait par interrogation du registre UDDI afin de trouver le service désiré et d'obtenir son URL. Des enregistrements peuvent êtres également faits auprès du registre UDDI afin de recevoir des notifications concernant des changements dans ce registre: ajout, retrait et modification au niveau des services.

2.4. Le Web sémantique

2.4.1. Définition et objectifs du Web sémantique

Le standard de description WSDL d'un service Web reste uniquement au niveau fonctionnel, c'est-à-dire qu'il contient la manière dont on peut utiliser le service et non ce que fait le service. Or, l'environnement du Web est ouvert, et les demandeurs de services peuvent êtres des machines plutôt que des personnes. Cela nécessite une description de services plus compréhensible et interprétable par ces machines afin de faciliter l'automatisation des processus d'enregistrement, de recherche, de sélection et d'acquisition des services Web requis et faire face à l'augmentation constante de leur nombre. Etant donné que la description WSDL est insuffisante pour mener, à terme, ces tâches, des travaux proposent des moyens de décrire des services Web en se basant sur les aspects sémantiques plutôt que syntaxiques.

Le Web sémantique est une vision qui permet de pallier ces difficultés et d'encoder les services Web dans une forme non ambiguë et compréhensible. Comme souligné dans **[72]**, l'objectif premier du Web sémantique est de définir et lier les ressources du Web afin de simplifier leur utilisation, leur découverte, leur intégration et leur réutilisation dans le plus grand nombre d'applications. Le Web sémantique doit fournir l'accès à ces ressources par l'intermédiaire de descriptions sémantiques exploitables et compréhensibles par des machines. Le Web sémantique doit aussi assurer de la manière la plus automatique qui soit, l'interopérabilité, la mise en œuvre de traitements et de raisonnements complexes sur les connaissances décrivant les ressources du Web tout en offrant des garanties élevées sur leur validité **[73]**. La description de ces connaissances repose sur l'emploi des ontologies. Une ontologie est définie dans **[74]** comme étant « *une spécification explicite d'une*

conceptualisation ». Une *conceptualisation* est un modèle abstrait qui représente la manière dont les personnes conçoivent les choses réelles dans le monde et une *spécification explicite* signifie que les concepts et les relations d'un modèle abstrait reçoivent des noms et des définitions explicites. Le Web sémantique est devenu un domaine à part entière, preuve en est la création en 2001 du groupe de travail sur ce sujet par le W3C **[75]**.

2.4.2. Architecture du web sémantique

Les travaux de recherche sur le web sémantique ont connu un grand développement après la standardisation du langage d'ontologie OWL (*Ontology Web Langage*) par le W3C **[76]**. Ce dernier est le résultat de la combinaison du langage de représentation d'ontologies DAML (*Darpa Agent Markup Language*) avec le langage d'inférence sur les connaissances ontologiques, OIL (*Ontology Inference Layer*) **[77]**. OWL se présente comme une suite logique des standards RDF et XML. Basé sur XML, RDF est le langage de marquage sémantique de données. Il constitue le langage de représentation des ontologies OWL. RDF et sa suite RDF-S offrent un modèle standard et simple de représentation des connaissances, qui peut être exploité automatiquement par une machine à travers des URI web **[78].** La **figure 2.7** illustre cette architecture en couches du web sémantique **[79]**.

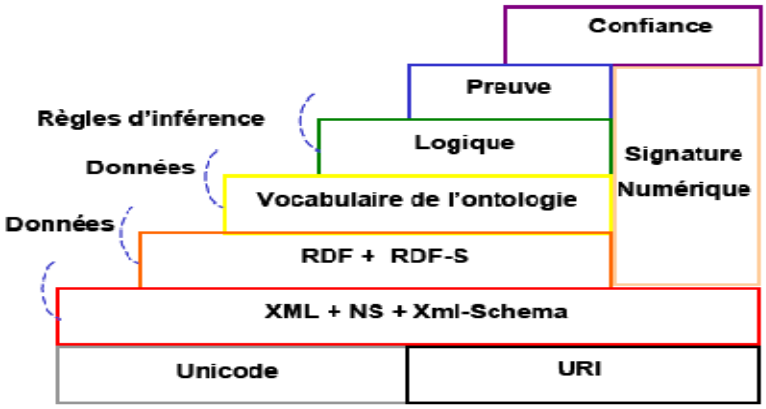


Figure 2.7 : Architecture en couches du web sémantique

RDF : RDF est un langage de représentation dont l'objectif est de permettre à une communauté d'utilisateurs d'utiliser les mêmes métadonnées pour l'accès aux ressources partagées **[78]**. Il a été conçu initialement par le W3C pour structurer l'information accessible sur le web et l'indexer efficacement. Le principal avantage de RDF est son extensibilité, à travers la réutilisation de schémas RDF issus d'autres documents et ce, grâce à l'utilisation du concept d'espace de nom « *namespace* ». Tout ce qui est exprimé avec RDF est appelé une ressource. Une ressource est toute entité d'information pouvant être référencée en un bloc, par un nom symbolique (littéral) ou un identificateur. L'identificateur est obligatoirement une URI. La syntaxe de base RDF est basé sur le triplet {propriété/ prédicat, ressource/ sujet, valeur/objet} qui peut être spécifié à l'aide du langage XML.

RDF-S : RDF-S (pour *RDF Schema*) a été proposé comme une extension sémantique de RDF. Ce langage permet d'associer aux propriétés RDF une sémantique avec la possibilité d'associer des contraintes sur les valeurs des propriétés **[80]**. Par exemple, si une propriété représente le nom d'une personne, on peut exiger que les valeurs de cette propriété soient une

référence à une personne et non pas à une voiture ou à une maison. On peut aussi restreindre les propriétés s'appliquant à une ressource. Par exemple, cela n'a probablement aucun sens d'autoriser une propriété « date de naissance » à être appliquée à un morceau de musique.

DAML-OIL : DAML+OIL est un langage d'ontologies qui a été proposé pour répondre aux utilisateurs désirant plus de facilités d'expressivité et de raisonnement que celles fournies par RDF et RDF-S sur les classes, les instances, les relations entre classes, les types, les énumérations, etc. La dernière version de DAML+OIL, intègre le typage de données basé sur le langage XSDL (*XML Schema Definition Language*) **[77]**.

OWL : OWL est un langage basé sur DAML-OIL et les logiques de descriptions (LD), le langage OWL offre plus de flexibilité pour la définition de classes et de relations complexes, en utilisant plusieurs types de constructeurs (intersection, union, restrictions, transitivité, cardinalité etc.) **[81]**. En s'appuyant sur une logique de description, OWL possède une sémantique formelle claire, ce qui permet de le doter de mécanismes de raisonnement (classification, inférence, vérification de cohérence). Actuellement, il existe un large panel d'outils de construction d'ontologies comme Protégé et OilEd. Ces outils sont associés à des moteurs d'inférence tels que Fact++, Racer ou Pellet. OWL se décline en trois sous langages avec des degrés d'expressivité croissants pour répondre aux besoins spécifiques des concepteurs d'ontologies **[76]**.

- OWL-Lite est destinée à des applications ayant besoin de décrire une hiérarchie de classification et de contraintes simples. Il peut être vu comme un sous ensemble réduit de structures OWL.
- OWL-DL est destinée à des applications nécessitant une expressivité maximale avec des restrictions particulières telles qu'une classe ne peut pas être une instance d'une autre classe. OWL-DL permet de garantir aux applications la décidabilité d'inférence.
- OWL-FULL est la version complète du langage OWL. Elle est destinée à des applications exigeant une expressivité maximale. Comparativement à OWL-DL, elle n'offre aucune garantie de décidabilité sur les inférences.

2.4.3. Les services Web sémantiques

2.4.3.1. Définition et intérêt des services Web sémantiques

Les services Web issus des travaux du domaine du Web sémantique sont appelés des *services Web sémantiques* (par opposition à l'appellation de service Web classique). D'après **[82]**, les services Web sémantiques sont la combinaison de la technologie des services Web et celle du Web sémantique. Ainsi, la description des services Web sémantiques est améliorée par des langages empruntés au Web sémantique, tel que RDF et OWL. Cet emprunt au Web sémantique permet à ces services Web d'être découverts et sélectionnés automatiquement par des machines ou d'autres services Web distants. Ceci permet aux services Web sélectionnés de répondre au mieux à la requête du client **[75]**. D'après **[83]**, les services Web sémantiques se situent entre deux problématiques liées au Web : l'interopérabilité et l'annotation sémantique des ressources **(Figure 2.8)**.

L'intérêt de combiner le Web sémantique et les services Web est de proposer une représentation intégrant les valeurs fournies par les services sous une forme compréhensible et interprétable automatiquement par les machines. Cette intégration permet ainsi l'automatisation des processus d'enregistrement (action d'identification du service Web), de recherche (action issue de la requête du client), de sélection (action de choix), de composition (action de combinaison), et d'exécution des services Web.

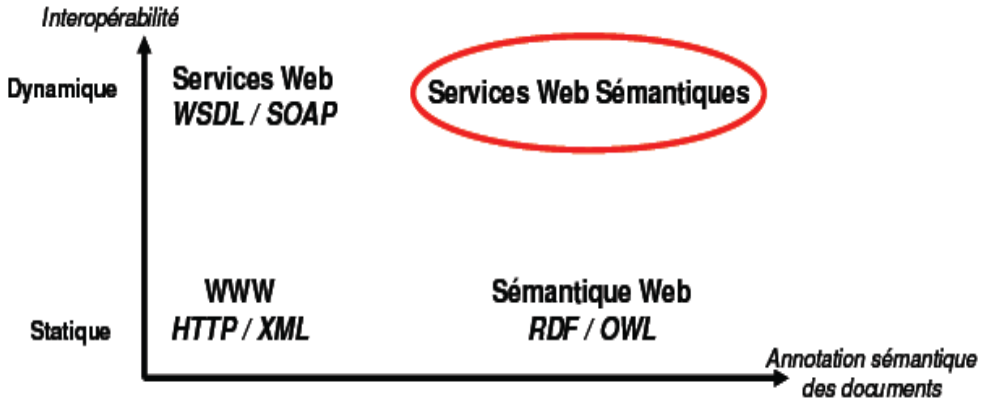


Figure 2.8 : Les services Web sémantiques **[83]**.

La description des services Web sémantiques repose sur deux choix. Tout d'abord, le choix du langage de représentation de connaissances à utiliser. Ensuite, le choix des concepts et relations à prendre en compte dans la description et leurs significations. Ceci implique la création d'ontologies ou la sélection d'ontologies existantes qui puissent fournir un vocabulaire ontologique structuré, c'est-à-dire un ensemble de concepts et de relations qui puissent être utilisés pour décrire des entités dans le domaine de la valeur **[84]**.

2.4.3.2. Langages de description des services Web sémantiques

Des travaux menés par W3C tentent d'apporter une solution standard en termes de description de services Web sémantiques. Deux langages sont issus de ces travaux: OWL-S et SAWSDL. Toutefois, aucun de ces langages ne s'est imposé comme une solution de description de services Web sémantiques.

OWL-S : OWL-S **[85]** (*Ontology Web Language for Services*) est un langage issu des travaux de la DARPAet de son programme *Agent Markup Language* (DAML) et prend la suite de DAML-S (*DARPA Agent Markup Language Service)*. Il a été intégré au consortium W3C en 2004, au sein du groupe d'intérêt sur les services Web sémantiques, lors de la recommandation du langage OWL. Le but initial du langage OWL-S est de mettre en œuvre des services Web sémantiques. Cette mise en œuvre inclut un grand nombre d'objectifs, rendus possibles par le biais de l'expressivité héritée d'OWL et de l'utilisation de la logique de description **[86]**. Le langage OWL-S organise la description d'un service en trois zones conceptuelles à savoir: *ServiceProfil*, *ServiceModel* et *ServiceGrounding* (**Figure 2.9**). Tout d'abord, le *ServiceProfile* est utilisé lors de la publication et la recherche d'un service. Il inclut trois types d'informations (décrites au format RDF): le *fournisseur*, le *comportement fonctionnel* et les *attributs fonctionnels*. L'élément *fournisseur* fourni une liste d'informations permettant d'entrer en contact avec le fournisseur du service Web (telle que son adresse

physique, l'URL du site Internet, son nom, son téléphone, etc.). L'élément *comportement fonctionnel* permet de rendre publiques les opérations disponibles que propose le service, ainsi que leurs paramètres d'entrée et de sortie. De plus, cet élément inclut la description des pré-conditions nécessaires à l'appel des opérations et les effets attendus à la suite de l'exécution du service. L'élément *attributs fonctionnels* apporte des informations supplémentaires concernant le service (telle que le coût du service, sa localisation géographique, sa disponibilité) afin qu'il soit sélectionné par de futurs clients.

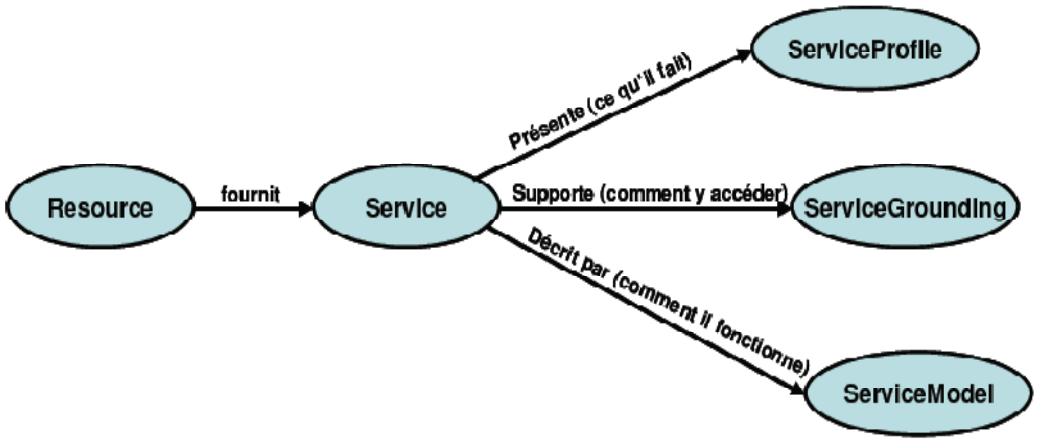


Figure 2.9 : Diagramme fonctionnel de l'ontologie de service **[85]**

Ensuite, le *ServiceModel*, dans le contexte d'un service Web élémentaire, présente le fonctionnement du service et définit comment interagir avec ce dernier. Dans le cadre de la description d'une composition de services, le *ServiceModel* permet de décrire l'activité du service dans une composition. Dans cette entité d'OWL-S, le service est vu comme un processus (*Process*). Un processus est une spécification de la manière dont le client peut interagir avec le service. Il est composé d'un ensemble d'informations sur ce dernier : son nom, les participants à son exécution (un client simple ou d'autres services Web), ce qu'il fait, les conditions d'utilisation et ses effets, son résultat, ses paramètres (entrée, sortie). Il existe trois types de processus : atomique (qui possède un accès direct), simple (qui fournit une vue sur un autre processus), et composé (qui est décomposable en processus simple). Un processus atomique est un processus qui peut être directement invoqué par l'intermédiaire d'un accès (utilisation direct du *GroundingProfile*). Enfin, Le *ServiceGrounding* apporte une description concrète de l'accès au service (protocole d'accès au service, format de messages, appel aux opérations et type de transport). Pour décrire ces informations, le *ServiceGrounding* fait référence à la description standard du service en WSDL. Le *ServiceGrounding* permet de concrétiser la définition du processus décrite dans le *ServiceModel*.

SAWSDL : SAWSDL (*Semantic Annotations for WSDL and XML Schema*) **[87]** est une recommandation du W3C depuis août 2007, et ce, à l'initiative du groupe de travail d'annotations sémantiques pour WSDL (*Semantic Annotations for WSDL Working Group*). SAWSDL représente la suite de WSDL-S (*Web Service Description Language – Semantic*) **[88]**. Son objectif étant d'ajouter de la sémantique à la description WSDL des services et des schémas XML **[89]**. Pour ce faire, SAWSDL définit un mécanisme d'annotation en vue de spécifier les éléments de WSDL à l'aide d'ontologies. Cette annotation repose sur la

définition d'attributs étendant le standard de description WSDL. Les annotations sémantiques référencent des ontologies préexistantes. Le mécanisme d'annotation de SAWSDL est indépendant de tout langage de représentation d'ontologies. SAWSDL propose deux sortes d'annotations sémantiques : une première pour identifier le concept sémantique (représentée par l'attribut *modelReference*) et une seconde pour faire le lien entre le concept et le document WSDL (représentée par les attributs *liftingSchemaMapping* et *loweringSchemaMapping*). L'attribut *modelReference* permet d'annoter tous les éléments de WSDL. Cependant, la recommandation de SAWSDL considère que l'annotation des éléments suivants : *interface, operation, fault* et *type*, apporte une signification particulière au service **[75]**. L'annotation de l'élément *interface* apporte une définition sémantique de l'ensemble des méthodes proposées par le service. Tandis que l'annotation de l'élément *operation* permet d'expliciter les effets des méthodes proposées par le service. L'annotation de l'élément *fault*, quant à elle, donne un sens au message d'erreur reçu lors de l'appel d'une méthode de l'interface. Enfin, l'annotation sémantique de l'élément *type* apporte une définition sémantique des paramètres échangés dans les messages. Ces paramètres sont décrits dans des structures de données qui peuvent être simple ou complexe.

2.5. Les services web pour dispositifs

2.5.1. De SOA vers SOAD

Les architectures SOA ont évoluées aux SOAD (Architectures Orientées Services pour Dispositifs) afin de mieux s'adapter aux caractéristiques des dispositifs et représenter l'ensemble des fonctionnalités fournies par ces dispositifs comme des services. On parle de services web pour dispositifs lorsque les technologies du Web sont utilisées pour implémenter les SOAD. Seules deux implémentations des services web pour dispositifs existent aujourd'hui : UPnP et DPWS. Les services web pour dispositifs présentent un ensemble de couches techniques génériques à savoir : l'adressage, le nommage, la découverte, la description, l'invocation, la notification et des couches techniques dites non-fonctionnelles. Toutefois, l'apport majeur des services Web pour dispositifs aux services web classiques concerne trois aspects principales : (1) la découverte de services, (2) la description de services et (3) les types de communications.

2.5.2. Mécanismes de découverte de services

La découverte est la première étape de la création d'applications à partir d'une infrastructure de services. C'est d'autant plus le cas lorsque les environnements dans lesquelles elles seront créées ne sont pas connus à priori. Toutefois, dans le paradigme de l'orientation service classique, la découverte repose sur la connaissance d'un type de service, ou d'un ensemble de fonctions, partagée par le fournisseur de service (le dispositif) et le demandeur de service (l'application détectant le dispositif). Cette connaissance étant publique car elle n'est pas associée à un seul dispositif. Dans ce schéma d'interaction, un fournisseur de service publie les interfaces de ses services auprès d'un registre de service et un demandeur de service peut demander au registre la liste des fournisseurs répondant à certains critères de sélection. Une évolution des SOA apportée par les SOAD est la modification de la découverte, pour prendre en compte des fournisseurs et consommateurs inconnus à l'avance.

Les entités qui seront présentes pour créer une application ne sont donc pas connues, et doivent être découvertes de façon dynamique. SOAD propose un nouveau schéma dynamique et décentralisé pour la découverte de services. Ce schéma repose notamment sur un modèle de collaboration utilisant les moyens multicast et broadcast répandus sur les réseaux locaux.

2.5.2.1. Moyens multicast et broadcast

Dans les réseaux locaux, le protocole IP Broadcast permet d'envoyer un message à toutes les entités connectées à un même réseau, tandis que le protocole IP Multicast ne vise que les entités ayant joint au préalable un groupe multicast donné. Afin de garder la fonctionnalité de découverte mais d'éviter le déploiement d'un répertoire de services, certaines solutions utilisent ces moyens de communication multicast et broadcast qui prennent la place logique du répertoire de services comme le montre la **figure 2.10**.

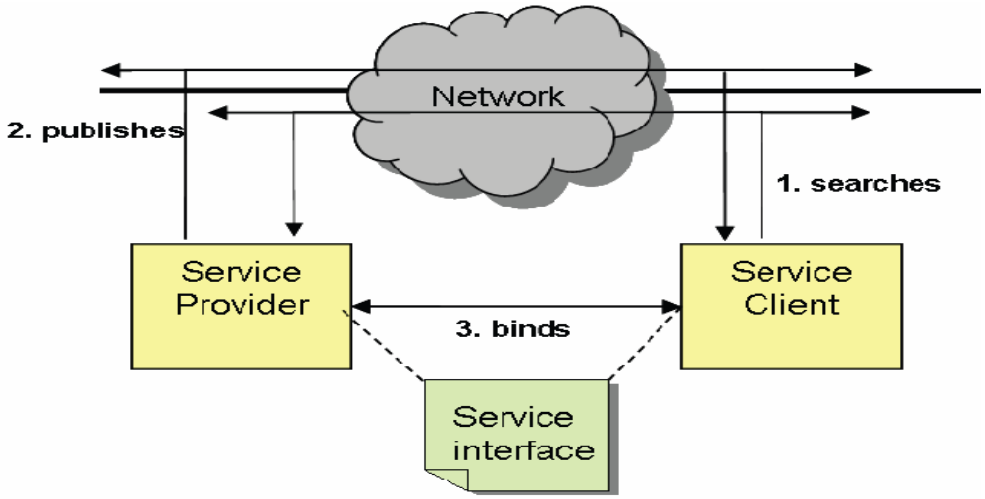


Figure 2.10 : Architecture Orientée Services avec utilisation de moyens multicast **[45]**

Les fournisseurs de services publient leurs services par annonce multicast ou broadcast sur le réseau. Les clients récupèrent de cette annonce les localisations des fournisseurs pour y accéder. Au contraire d'une annonce broadcast qui est écoutée par tous les clients du réseau, une annonce multicast est écoutée uniquement par les clients qui se sont joints au réseau multicast défini par le protocole. Les clients peuvent, à leur tour, envoyer une requête par diffusion (multicast ou broadcast) afin de connaître tous les fournisseurs de services. Les fournisseurs répondent alors seulement au client (unicast) qui a effectué la recherche, en indiquant les informations pour les contacter. La requête du client peut être aussi raffinée par des moyens et des critères de sélection de service. Chaque fournisseur de service correspondant aux critères de sélection répond alors par un message de réponse unicast à l'adresse du requérant.

2.5.2.2. Répertoires de service

Les protocoles IP Multicast et Broadcast passent difficilement à l'échelle de grands réseaux. En effet, ces protocoles chargent la bande passante et les moyens de routage. De plus, ces protocoles sont contre-indiqués dans les réseaux de communications sans-fil tels que Bluetooth, WiFi ou Infrarouge **[90]**, car les multicast ou broadcast sont souvent gourmands en énergie et en bande passante. Une architecture utilisant un ou plusieurs répertoires de services

est une solution qui permet le passage à l'échelle des protocoles de découverte utilisant IP Multicast et Broadcast. Dans cette architecture, les moyens de diffusion (multicast ou broadcast) sont utilisés pour découvrir un registre de services. Ce dernier permet alors aux consommateurs (clients) d'avoir un seul interlocuteur et d'effectuer des requêtes en point à point (unicast). Un tel registre construit sa base de données à partir des messages d'annonce des fournisseurs. Ainsi, en établissant des moyens unicast sans communication de groupe, un répertoire de service fiabilise et économise les messages liés à la découverte sur les réseaux locaux. Plusieurs répertoires de services peuvent être déployés par redondance, par spécialisation ou par hiérarchisation selon les exigences techniques visées par un protocole : fiabilité, passage à l'échelle et séparation de domaine d'administration **[45]**. La redondance augmente la fiabilité par le déploiement de plusieurs répertoires identiques. Dans ce cas de figure, les fournisseurs de services doivent enregistrer leurs services dans tous les répertoires découverts. La spécialisation de chaque répertoire déployé dans un domaine d'administration donné assure le passage à l'échelle et la séparation de domaine d'administration. La hiérarchisation de répertoire permet, quant à elle, le passage à l'échelle de grands réseaux tels qu'Internet.

2.5.2.3. Découverte active et découverte passive

Deux types de mécanismes de découverte de services sont à distinguer : les mécanismes de découverte active et les mécanismes de découverte passive **[45]**. Dans les mécanismes de découverte active, les clients (demandeur de service) sont forcés de renouveler leurs requêtes afin de connaître la disponibilité des fournisseurs de services. Ces requêtes sont envoyées à l'adresse du répertoire de service ou à une adresse multicast. Les réponses sont alors reçues par le demandeur de manière synchrone de la part du répertoire de service ou directement de la part de tous les fournisseurs correspondant à la requête active. Les mécanismes de découverte passive, quant à eux, permettent de notifier les clients des départs et des arrivées des fournisseurs de service de manière asynchrone. Ces mécanismes permettent au demandeur de service d'écouter les publications spontanées de services selon deux modes différents: mode de souscription aux événements de publications auprès du registre de services et mode d'écoute passive aux annonces d'une adresse multicast. Dans le premier mode, les souscriptions des demandeurs de service sont gardées en mémoire du répertoire de services comme des requêtes persistantes. Le répertoire de services envoie alors une réponse de manière asynchrone à un demandeur dès qu'un service publié correspond à sa requête. Dans le second mode, les fournisseurs de services publient spontanément leurs services par les moyens multicast. Les demandeurs de services doivent alors se joindre à l'adresse multicast afin de recevoir passivement les annonces.

Pour des raisons d'efficacité et de fiabilité, la plupart des protocoles de découverte définisse les deux mécanismes de découvertes. En effet, lorsqu'un client arrive dans l'infrastructure réseau, c'est à lui d'initier la découverte active pour connaitre la liste des fournisseurs de services présents. Par la suite, le client peut écouter passivement les annonces spontanées des fournisseurs à leur connexion et leur déconnexion. Les deux modes de découvertes sont donc utiles. Par ailleurs, du fait de la possibilité d'erreurs de communication, la répétition des requêtes actives et des messages de notification spontanée est souvent recommandée.

2.5.3. Description de services

La phase de description est nécessaire dans les Architecture Orientée Service pour Dispositifs où les clients et serveurs ne se connaissent pas au préalable. A l'instar des SOA, SOAD décrit les entités à l'aide d'interfaces de service. Cette description contient une liste de méthodes (opérations) invocables pour réaliser le service en question, parfois aussi une liste de variables qui renseigne sur l'état du service, et éventuellement une description textuelle des services. Cette interface peut être utilisée, par exemple, pour générer le code qui communiquera avec le fournisseur ou pour créer un composant réutilisable dans une plate-forme d'assemblage **[91]**. Une interface d'un service web pour dispositif mentionne généralement les éléments suivants **[45]**: les *espaces de noms*, les *types*, les *messages*, les *opérations*, et les *événements*. Les *espaces de noms* "*namespaces*", à l'image des packages en Java, permettent à une interface d'importer des éléments (types, messages, etc.) depuis d'autres interfaces ou spécifications. Les *types* font référence aux types des arguments des opérations. Les types peuvent être simples (entiers, chaines de caractères, booléens, etc.) ou des structures de données complexes. Les *messages* sont définis par une liste ordonnée de variables (arguments) typées qui peut être échangées entre un client et un serveur. Les *opérations* sont appelées de différentes manières selon les technologies (eg. UPnP les appellent "actions"). Une opération est décrite par son nom dans l'espace de nom de l'interface, par un message d'entrée ou un message de retour ou par les deux types de messages. Quatre types génériques d'opérations se distinguent : *request-response*, *one-way*, *solicit-response* et *notification.* L'opération *request-response* nécessite deux types de messages: le client envoie un message de requête, le serveur répond par un message de réponse. Ce mode opératoire est le plus commun. Le mode opératoire *one-way* dérive du mode *request-response*. La différence est qu'aucun message de réponse n'est attendu après réception et traitement de la requête par le serveur. Le mode opératoire *solicit-response* est un mode inverse au mode *request-response*. Le serveur prend ici l'initiative d'envoyer un message au client auquel celui-ci répond par un message. Ce mode nécessite la souscription préalable du client sur les événements adéquats du serveur. Le mode opératoire *notification* dérive du mode *solicit-response*. La différence est qu'aucun message de réponse n'est attendu par le serveur après notification du client. Il nécessite de même la souscription préalable du client sur les événements adéquats du serveur. Ce mode de notification est répandu dans la plupart des intergiciels de communication réparti au contraire du mode *solicit-response* **[45]**. Les *événements* correspondent aux envois de messages spontanés par le serveur au client, soient aux opérations de type *solicit-response* et *notification*. L'émission d'événements d'un serveur vers le client nécessite la souscription préalable du client à ceux-ci.

2.5.4. Types de communications

2.5.4.1. Messages synchrones : Invocations ou contrôle

Dans ce type de communication, les opérations d'un service sont appelées par le client via un appel distant selon un protocole de communication (SOAP, RMI, etc.). Ce schéma de communication correspond à l'utilisation classique d'un service dans lequel les messages sont synchrones : un consommateur envoie une requête au fournisseur, qui lui retourne un message lorsque l'action est effectuée. Les services pour dispositifs spécifient le contenu des en-têtes

et du message utile (*payload*) pour les messages d'entrée, les messages de retour et les messages d'erreur. Les messages de retour contiennent les informations demandées par la requête et les messages d'erreur permettent de vérifier que l'action a été bien effectuée. D'une façon générale, ce type de communication permet, d'une part de demander des informations à un dispositif et d'autre part d'invoquer des actions afin de prendre le contrôle de ce dispositif et éventuellement changer son état.

2.5.4.2. Messages asynchrones : Notification ou événements

Imaginons qu'un client souhaite connaitre le changement d'état d'un dispositif particulier. Avec la communication synchrone, le client est obligé de demander périodiquement au dispositif s'il a changé d'état. Ce type de communication s'avère ainsi couteux et inefficace pour ce genre d'utilisation. Plutôt que de réaliser une attente active (*polling*) par le client, les services pour dispositifs fournissent un autre type de communication asynchrone basé sur la notification. La notification recouvre les opérations de type *notification* et *solicit-response*. Dans les deux cas, l'envoi de message est initié par le serveur (dispositif) et destiné au client. Le contrôle est ici inversé et requiert une opération préalable: la souscription (ou abonnement) du client à des événements du serveur. Cette souscription peut s'effectuer selon deux modes différents : *observation classique* et *publish/subscribe* (publication/souscription) **[45]**.

- **Observation classique :** dans ce modèle classique, deux interfaces de services sont normalisées: l'observé et l'observateur. L'interface "observé" définit une opération de souscription à un ou plusieurs événements. Le serveur implémente cette opération qui sera appelée par le client pour montrer son intérêt à certains événements. La souscription est valide pour une durée déterminée ou jusqu'au moment où l'opération de retrait de souscription est appelée. L'interface "observateur", compte à elle, définit une opération de notification. Le client implémente cette opération qui sera appelée par le serveur lors de la détection d'événements intéressant le client.
- **Publish/Subscribe :** Le paradigme *Publish/Subscribe* définit un mécanisme plus élaboré. Ce mécanisme nécessite au moins un acteur supplémentaire qui hébergera des "sujets" ("topics"). Les serveurs publient leurs événements auprès des sujets pertinents. Les clients s'abonnent aux sujets qui les intéressent. Chaque sujet est responsable de transmettre les événements publiés à la liste des abonnés. Ce paradigme induit un couplage très lâche entre clients et serveurs. En effet, ces entités ne se connaissent pas si elles n'interagissent que par ce mécanisme. Les clients et serveurs ne connaissent que les sujets. Seuls les sujets ont une connaissance des clients intéressés et des serveurs d'événements correspondant.

2.6. L'orientation service pour l'intelligence ambiante

2.6.1. Adéquation de l'orientation service pour les environnements ambiants

Comme nous l'avons vu dans **la section 2.2.2**, les intergiciels distribués usuels reposent sur la conception conjointe de clients et de serveurs. Ces derniers partagent un même langage de programmation, un même protocole et une connaissance mutuelle de leur implémentation. De plus, ces clients et serveurs sont administrés de manière commune par une même entité.

Cela implique que les environnements de déploiement des clients et des serveurs doivent être statiques et homogènes. Cette vision s'oppose par conséquent à la vision de l'intelligence ambiante qui décrit des environnements ouverts où les applications visées devront composer des entités hétérogènes qui ne se connaissent pas à l'avance et réagir dynamiquement à l'arrivée et au départ de nouvelles entités dans l'environnement.

L'orientation service répond particulièrement à l'ouverture et à la dynamique des environnements ambiants. Dans ce contexte, les architectures orientées services (SOA) sont utilisées pour représenter l'ensemble des fonctionnalités fournies par les dispositifs de l'environnement comme des services. L'utilisation des services pour représenter les dispositifs offre des facilités pour l'interopérabilité entre les dispositifs et pour la découverte de services. Elle fournit également la dynamicité nécessaire aux applications ambiantes par la gestion de l'apparition et la disparition de dispositifs physiques comme l'arrivée et le départ de services. En effet, les SOAD apportent une telle évolution aux SOA. A travers notamment ses mécanismes d'annonces et de communication par événements, les entités sont capables de connaître les entités se trouvant dans le réseau et les voir apparaître et disparaître dynamiquement. Ces mécanismes ajoutent alors une dynamicité aux interactions entre les dispositifs prenant part aux applications ambiantes.

En outre, grâce notamment aux mécanismes de couplage lâche entres les services, l'architecture SOA permet l'intégration des services implémentés dans des langages de programmation différents et fournis par des fournisseurs distincts. SOA permet aussi une séparation claire entre l'administration des services fournis et celle de l'application construite à partir de ces services. En effet, les services proprement dits sont administrés par leurs fournisseurs respectifs. Ces derniers enregistrent leurs services auprès d'un répertoire de services. Ils tiennent leurs enregistrements à jour et les suppriment s'ils ne les fournissent plus. Les applications, compte à elles, sont construites grâce à des liaisons temporaires entres les services. Des mécanismes de découvertes de services sont alors nécessaires afin de tenir ces applications à jour sur la disponibilité de ses services constitutifs. Deux types de mécanismes de découverte de services sont à distinguer : les mécanismes de découverte passive et les mécanismes de découverte active. Les mécanismes de découverte passive permettent de notifier les clients des départs et des arrivées des fournisseurs, contrairement aux mécanismes de découverte active où les clients sont forcés de renouveler leurs requêtes afin de connaître la disponibilité des fournisseurs de services. Dans les deux cas, la dynamique du répertoire de services permet une gestion réactive des applications clientes à la disponibilité des fournisseurs.

Les concepts d'architecture orientée service sont appliqués dans le domaine de l'informatique pervasive et ambiante par de nombreux travaux, par exemple **[45] [92] [93]**. Dans ces travaux, les équipements sont modélisés comme des fournisseurs et des demandeurs de services. Certains équipements demandeurs de services composeront les services disponibles sur le réseau afin de fournir d'autres services de plus haut niveau. La **table 2.2** présente notre mapping des concepts de l'intelligence ambiante vers l'approche SOA. Dans ce mapping, toutes les fonctionnalités fournies par les dispositifs sont vues comme des services. Ainsi, tous les dispositifs de l'environnement ambiant sont considérés comme des fournisseurs de services. L'annonce et la recherche des fonctionnalités par les dispositifs

ambiants correspondent respectivement à la publication et la découverte des services dans le contexte de SOA. Par ailleurs, les utilisateurs de l'environnement ambiant qui sollicitent des fonctionnalités sont considérés comme des demandeurs de services. La tâche demandée par l'utilisateur peut être spécifiées sous forme d'une requête et d'un événement. L'application réalisant cette tâche correspond à un service composite dans le contexte de SOA. Ainsi, la réalisation des tâches de l'utilisateur en composant les fonctionnalités disponibles dans l'environnement ambiant se traduit par la composition dynamique de services en SOA. En outre, l'identification des fonctionnalités pertinentes et leur mise en correspondance se traduisent respectivement par la selection et le *matching* des services en SOA. Enfin, la sensibilité au contexte se traduit par la sensibilité à la qualité de service fournie, tandis que l'adaptation de l'application correspond au monitoring de services en SOA.

Intelligence ambiante	**Architecture Orientée Service (SOA)**
Dispositif	Fournisseur de services
Fonctionnalité	Service
Utilisateur	Demandeur de services
La tâche de l'utilisateur	Requête ou événement
Application réalisant la tâche de l'utilisateur	Service composite
Annoncer les fonctionnalités	Publication de services
Rechercher des fonctionnalités	Découverte de services
Identifier les fonctionnalités pertinentes	Sélection de services
Correspondance entre les fonctionnalités	*Matching* de services
Réalisation de la tâche de l'utilisateur	Composition dynamique de services
Sensibilité au contexte	Sensibilité à la qualité de service (QoS) fournie
Adaptation de l'application	Monitoring de services

Table 2.2 : Environnement ambiant orienté services.

2.6.2. Modèles orientés services pour les exigences de l'intelligence ambiante

L'adoption de l'approche orientée service pour modéliser les systèmes intelligents ambiants doit fournir des réponses concrètes aux différentes exigences de ces systèmes. Comme nous l'avons détaillé dans la **section 1.4 du chapitre 1**, les systèmes AmI sont construits autour de six exigences fondamentales à savoir : l'intelligence, la sensibilité au contexte, l'auto-adaptation, l'intéropérabilité, la transparence et le passage à l'échelle. L'approche orientée service doit ainsi fournir des modèles et des mécanismes nécessaires afin de tenir compte de chaque exigence. Dans les sections suivantes, nous présentons les modèles adéquats en orientation service pour la réalisation des exigences de l'intelligence ambiante.

2.6.2.1. Réalisation de l'intelligence

L'intelligence d'un système ambiant réside dans sa capacité de construire dynamiquement des applications afin de répondre aux besoins des utilisateurs. Dans le contexte de SOA, cette intelligence peut être concrètement réalisée grâce à trois modèles essentiels, à savoir la composition de services, le *matching* de services et la découverte de services.

Modèle de composition de service : le modèle de composition de services essaye de fournir les fonctionnalités demandées (requête) en combinant les fonctionnalités disponibles annoncées par les fournisseurs de services et découvertes par le modèle de découverte des services. Le modèle de composition génère un ou plusieurs services composites selon diverses techniques de composition. Ces techniques sont principalement différentes en termes de leur degré d'automatisation. Selon le degré d'intervention humaine pour la spécification du service composite, on peut distinguer les approches manuelles et automatiques. Dans les approches manuelles, le service composite est spécifié soit par le développeur ou bien par l'utilisateur. Dans le premier cas, le service composite est spécifié au moment du développement en utilisant certains outils destinés à la création des modèles généraux (structures) d'applications. Cela implique que l'utilisateur n'a aucun moyen d'interagir avec le service composite lors de son exécution. Dans le second cas, le service composite est spécifié par l'utilisateur au moment de l'exécution. Cela laisse une marge de manœuvre à l'utilisateur afin d'adapter l'application à ses besoins spécifiques. En effet, lorsque le développeur spécifie le service composite, il peut seulement essayer de prévoir les services qui seront disponibles au moment de l'exécution. Tandis que, l'utilisateur spécifie le service composite à partir des services réellement disponibles dans l'environnement au moment de l'exécution. Dans les approches automatiques, le service composite est généré par le système, et l'intervention humaine est réduite juste à la spécification des besoins (tâche) et des exigences de qualité de service.

Modèle de *matching* des services : il est à noter que, pour tout mécanisme de composition de services, la métrique de similarité fournit la clé d'une part, pour le regroupement des services fonctionnellement équivalent et d'autre part, pour *mapper* (correspondre) les services disponibles avec les tâches demandées. Les mesures de la similarité sémantique ont été largement étudiées dans le cadre du concept du *matching* sémantique des services ou le *matching* des services tout court. Le *matching* des services consiste à trouver des services qui fournissent des fonctionnalités (i.e., capacités) qui correspondent aux fonctionnalités demandées par la requête. En d'autres termes, c'est la comparaison, par paire, des services annoncés avec ceux souhaités (requête) afin de déterminer le degré de leur correspondance sémantique (*match* sémantique). Ce processus peut être non-basé sur la logique, à base de la logique ou hybride selon la nature du raisonnement utilisé. Le *matching* des services est principalement utilisé pour effectuer plusieurs processus comme la classification, la sélection et la découverte des services. Chaque processus peut intégrer le *matching* des services comme un module interne ou tout simplement l'invoquer comme un module externe à chaque fois que le besoin se fait sentir.

Modèle de découverte des services : la découverte des services est le processus de recherche et de localisation des services disponibles qui peuvent prendre part à une composition de

service. On peut distinguer deux types de découverte des services: une découverte globale des services et une découverte spécifique des services. Le rôle de la découverte globale des services consiste à collecter et maintenir un ensemble de tous les services disponibles dans l'environnement ambiant. Ce genre de découverte peut être réalisée, par exemple, par un moteur de recherche *front-end*, par des fournisseurs de services qui publient leurs services dans des registres ou par des agents intermédiaires. La découverte spécifique des services, quant à elle, son rôle consiste à rechercher des services particuliers qui répondent à une description bien spécifique des paramètres fonctionnels et non-fonctionnels. Cette découverte se produise généralement lorsque l'on cherche à réutiliser certaines fonctionnalités spécifiques des services existants dans la création d'un nouveau service composite. Les méthodes utilisées pour la découverte des services varient considérablement en fonction de l'environnement dans lequel les services évoluent. Pour un réseau local, une base de données centrale de tous les services mis en ligne pourrait être appropriée. Dans ce cas, la découverte est effectuée par le gestionnaire de la base de données qui maintient les descriptions de tous les services disponibles. Pour un réseau à large échelle, un système de recherche hiérarchique est utilisé pour une meilleure scalabilité.

2.6.2.2. Réalisation de la sensibilité au contexte

La sensibilité au contexte réside dans la capacité d'un système d'évaluer en permanence le contexte courant et d'en tenir compte pour mieux répondre aux besoins des utilisateurs. La sensibilité au contexte se traduit en SOA par l'évaluation des attributs de la qualité de service (*QoS)* et l'utilisation de ces attributs dans le processus de sélection de services.

Modèle d'évaluation de la *QoS* et du contexte : une fois présentés dans un modèle compréhensible, la *QoS* et le contexte devraient être évalués pour être utilisés dans le processus de sélection des services. Différents types d'attributs de la *QoS* et du contexte peuvent exister pour la même application, tels que: les attributs qualitatifs et quantitatifs, les attributs déterministes et non-déterministe, etc. En conséquence, plusieurs formules indiquant la façon dont les attributs sont calculés peuvent être établies en fonction du type de l'attribut lui-même. Par la suite, les différentes formules sont intégrées dans des fonctions d'utilité qui estiment la qualité globale espérée lors de l'exécution d'un service composite dans un contexte donné. A ce niveau, une pondération est attribuée à chaque formule en fonction de la pertinence de l'attribut qu'elle estime par rapport au contexte d'utilisation. Ainsi, de nombreux services composites qui constituent des configurations alternatives d'une même application peuvent être établis avec leurs valeurs d'utilité respective. Le Framework compare ainsi l'utilité obtenue de toutes les alternatives présentées et sélectionne celle qui fournit la meilleure valeur.

Modèle de sélection des services : quand plusieurs services similaires (i.e., exposent des fonctionnalités identiques ou similaires) sont disponibles, quel sera le service à intégrer dans le service composite qui est en cours de création? Il demeure difficile de trouver des services qui répondent parfaitement aux exigences des utilisateurs et qui soient en adéquation avec le contexte d'utilisation. Cette tâche est généralement déléguée au processus de sélection des services. Ce processus consiste à choisir le service le plus pertinent parmi une liste de services candidats fonctionnellement équivalents. La sélection des services dépend fortement

de l'évaluation de nombreux critères tels que le contexte d'utilisation et les propriétés non fonctionnelles des services (*QoS*). Les critères les plus utilisés pour effectuer ce choix sont le temps requis par un service pour une exécution complète et le cout généré par cette exécution. Le cout est un attribut générique qui peut englober : le taux d'occupation du CPU, l'espace mémoire requis, la consommation énergétique, etc. En général, le processus de sélection des services est effectué par un mandataire de service. Dans certains cas, le mandataire de service est désigné au préalable sur un ou plusieurs dispositifs prédéfinis. Dans d'autres cas, où tous les dispositifs ne sont pas identiques en terme de la puissance de traitement et aucune garantie n'est fournie sur la disponibilité d'un dispositif à un instant donné, une certaine forme de mise en candidature est alors utilisée. Dans ce cas, les dispositifs qui sont prêts à jouer le rôle de mandataire de service peuvent s'inscrire dans un registre centralisé. Par la suite, la sélection du dispositif est faite par le gestionnaire des enregistrements.

2.6.2.3. Réalisation de l'auto-adaptation

L'auto-adaptation reflète la capacité d'un système à gérer les changements de contexte afin de mieux s'adapter aux nouvelles situations. Dans le contexte de SOA, l'auto-adaptation est réalisée par des mécanismes de monitoring de services.

Modèle de monitoring des services : le monitoring d'un service composite consiste à gérer son cycle de vie depuis sa création jusqu'à son exécution finale. Le monitoring peut faire appel au modèle de composition de service lors de la phase création et au modèle de sélection des services lors de la phase exécution. Afin de garantir une meilleure exécution, le monitoring doit contrôler chaque service constituant le service composite. Ce contrôle se base sur la gestion des incertitudes et l'adaptation dynamique du service composite face aux changements imprévisibles du contexte d'utilisation qui peuvent se produire à tout moment. Tout dépend du degré des changements qui surviennent dans l'environnement ubiquitaire, l'adaptation dynamique peut aller du remplacement d'un service élémentaires jusqu'au changement de la structure complète du service composite. Ce changement de structure s'effectue par l'auto-configuration qui consiste à ajuster dynamiquement et d'adapter la configuration d'un service composite donné au contexte courant d'utilisation. Le monitoring des services se charge également de fournir les formats et les mécanismes nécessaires afin de faciliter le stockage et la réutilisation d'un service composite en cas de besoin. Enfin, le monitoring de services ajoute l'aspect dynamique à la découverte et la sélection de services :

- *Découverte dynamique de services*: permet la découverte et la localisation des services et des ressources pour répondre aux besoins d'un environnement volatile où les services peuvent apparaître et disparaître dynamiquement.
- *Sélection dynamique de services*: détermine le service le plus approprié parmi les services candidats qui fournissent des fonctionnalités similaires ou proches. Le but de la sélection dynamique des services est de répondre en permanence aux besoins des utilisateurs et d'améliorer la qualité de service fournie par l'anticipation des utilisations des ressources ou services futures et en tenant compte du contexte actuel.

2.6.2.4. Réalisation de l'intéropérabilité

L'interopérabilité peut étre réalisée par une couche d'abstraction qui se charge de fournir des modèles communs pour la représentation des connaissances et la gestion des communications.

Modèle des connaissances : il est clair que la composition de services exige un certain niveau d'expressivité dans la façon dont l'information est modélisée et présentée. C'est pourquoi un modèle des connaissances compréhensible devrait être fourni par chaque système dédié pour la composition de service. En général, un modèle des connaissances comprend trois éléments importants: le service, la *QoS* et le contexte. Tout d'abord, le modèle de service vise à décrire les attributs fonctionnels et non fonctionnels des services. Plusieurs langages de description des services (appelés également les formats de description des services) ont été développés. Ces langages proposent des niveaux sémantiques différents pour la spécification des services fournis et demandés pour une utilisation ultérieure dans le processus de composition. Ensuite, le modèle commun de la *QoS* permet d'exprimer toutes les dimensions de la qualité de service utilisées par le Framework afin de sélectionner et de déployer le service composite qui offre la meilleure qualité. Ce modèle facilite également le traitement et le raisonnement sur les attributs de la qualité de service. Dans de nombreuses approches proposées, la qualité de service est associée à des services comme paramètres non-fonctionnels. Enfin, comme le contexte est plus général que la qualité de service, le modèle du contexte prend en compte tous les aspects qui entourent une application tels que les utilisateurs, les dispositifs, l'environnement, etc. Dans de nombreux cas, la qualité de service est considérée comme une partie du contexte. Ainsi, les attributs de qualité de service qui sont pertinents pour une application deviennent une partie importante du contexte de cette application. Dans certains cas, le même attribut peut désigner une qualité et un contexte. Par exemple, l'attribut «*température*» peut se référer à la température à l'intérieur d'une salle (contexte), et peut se référer également à la température ambiante spécifiée par l'utilisateur (qualité). Par conséquence, si le contexte de la salle change, la qualité offerte à l'utilisateur change aussi. En général, les changements de contexte peuvent conduire à la dégradation de la qualité de service fournie. Par conséquent, le système doit s'adapter afin de maintenir une qualité de service satisfaisante dans le nouveau contexte. La *QoS* et le contexte peuvent être exprimés dans un même modèle ou séparément. Une ontologie est souvent utilisée pour exprimer la qualité de service et les paramètres contextuels.

Modèle de communication : l'approche de communication spécifie la topologie et la technologie de communication utilisées par le système. La topologie indique si la composition de service est effectuée d'une manière centralisée ou décentralisée. Certaines approches nécessitent un nœud central qui agit comme un coordinateur de composition de services et imposant de ce fait une structure centralisée. D'autres approches permettent une structure décentralisée ou hiérarchique. La technologie de communication, quant à elle, indique le standard utilisé afin de gérer la communication entre les différents dispositifs disponibles dans l'environnement ambiant. Cette technologie peut être la pile standard d'un service Web, un service Web pour dispositifs tels que l'UPnP et DPWS, ou tout autre infrastructure de bas niveau qui englobe et gère tous les aspects liés à la communication inter-dispositifs. La technologie utilisée devrait fournir un certain support de dynamisme puisque dans un environnement ambiant, les dispositifs peuvent ne pas être connectés à tout moment. Par conséquent, il n'ya aucune garantie quant à une connexion continue pendant l'utilisation d'un service et les dispositifs peuvent apparaitre et disparaitre de façon dynamique.

2.6.2.5. Réalisation de la transparence et du passage à l'échelle

La transparence d'un système dépend de son architecture générale qui décrit la façon dont ce système communique avec les utilisateurs et l'environnement externe. Cette transparence dépend également du degré d'automatisation des différents mécanismes intervenant dans l'architecture du système. La performance du passage à l'échelle, quant à lui, dépend des modèles d'évaluation adoptés par le système.

Architecture générale : l'architecture générale d'un système permet de mieux comprendre son fonctionnement global à travers notamment les fonctionnalités offertes ainsi que la conception globale et l'implémentation du système. L'architecture générale permet également de décrire les aspects internes et externes d'un système. L'aspect interne décrit les différents modules qui constituent le système et les relations entre eux. Tandis que l'aspect externe décrit les interactions du système avec les utilisateurs et l'environnement via une infrastructure de communication commune. En outre, cette partie fournit un aperçu général sur le processus de composition de service commençant par la création d'un service composite jusqu'à son exécution finale. L'architecture générale semble être le meilleur moyen de comprendre un système avant de détailler ses modules internes. Le degré d'automatisation de ces modules dépend du degré de participation des utilisateurs dans leur spécification et fonctionnement. Une automatisation élevée réduit considérablement l'intervention humaine et incrémente ainsi la transparence du système.

Modèle d'évaluation : l'objectif principal de l'approche d'évaluation adoptée par un système est de démontrer ses performances et sa faisabilité dans un environnement ambiant réel. La plupart des systèmes proposés sont évalués soit par des exemples simples ou par des scénarios applicatifs. Les exemples d'applications servent généralement à illustrer une large variété des utilisations possibles des fonctionnalités du système, tandis que les scénarios applicatifs mettent l'accent sur des situations réelles dans lesquelles le système se mette en coopération avec les utilisateurs et l'environnement afin de résoudre des problèmes pertinents. Toutefois, une approche d'évaluation ne devrait pas se limiter uniquement aux PC de bureau traditionnels mais doit tenir compte d'une large gamme de dispositifs mobiles et distribués dans l'environnement ambiant tels que les robots, les capteurs, les actionneurs, etc. Une approche d'évaluation peut également montrer des tests expérimentaux des performances des différents mécanismes proposés en relation avec la composition de service.

2.6.3. La composition de services au cœur des modèles orientés services

Comme nous l'avons vu dans la **section 2.6.2**, un système AmI nécessite de tenir compte de plusieurs aspects en relation directe avec la composition de services. C'est ainsi que la problématique de composition de services se situe au cœur des différents modèles orientés service. Tout d'abord, la composition de service exige une description standard et compréhensible de toutes les connaissances manipulées. Le modèle des connaissances devrait inclure des descriptions pour les services, les événements ou les tâches, la qualité de service (*QoS*) et le contexte d'utilisation. Une fois le modèle des connaissances est spécifié, la deuxième étape dans la création d'un service composite consiste à découvrir et localiser les services qui fournissent les fonctionnalités requises par le nouveau service composite. Dans un certain nombre d'approches proposées, la découverte des services est basée sur les

méthodes de *matching* et de sélection des services. Le *matching* des services consiste à trouver des services qui fournissent des fonctionnalités (i.e., capacités) qui correspondent aux fonctionnalités demandées. La sélection des services, quant à elle, consiste à choisir un ou plusieurs services parmi ceux retournés par le processus de *matching*. Le processus de sélection est essentiellement basé sur l'évaluation des paramètres non-fonctionnelle notamment la *QoS* et le contexte, afin de fournir un service composite avec la qualité requise et en adéquation avec le contexte d'utilisation. La troisième étape concerne le modèle de composition lui-même qui fournit les fonctionnalités demandées par la combinaison de plusieurs services disponibles dans l'environnement ambiant. Pour ce faire, plusieurs techniques peuvent être utilisées et, par conséquent, un ou plusieurs services composites (i.e., plans de composition) peuvent être générés. La quatrième étape est le monitoring des services composites. Le monitoring consiste à fournir des méthodes et des mécanismes pour supporter l'exécution des services composites en garantissant une adaptation dynamique de leurs structures face aux changements continuels du contexte. La cinquième étape consiste à spécifier l'infrastructure technologique qui est utilisée comme modèle de communication entre les différents dispositifs physiques et entités logiques de l'environnement ambiant. Le modèle de communication indique également si la composition est réalisée de manière centralisée ou décentralisée. La sixième et dernière étape porte sur l'approche adoptée pour l'évaluation du système. Cette étape vise l'évaluation expérimentale des performances des différentes approches proposées pour supporter la composition de service à large échelle. Elle vise également la validation du système sur des scenarios réels d'utilisation.

2.7. Conclusion

Dans ce chapitre, nous avons souligné les caractéristiques importantes des environnements ambiants à savoir : l'ouverture, l'hétérogénéité, l'incertitude et la dynamique des entités constituants ces environnements. Ces caractéristiques imposent des défis scientifiques considérables pour la mise en place des systèmes intelligents ambiants. Les principaux défis soulevés sont: l'intelligence, la sensibilité au contexte, l'auto-adaptation, la couche d'abstraction, la transparence et le passage à l'échelle. Face à ces défis, le paradigme de l'orientation service apporte l'interopérabilité et le couplage lâche nécessaire à la construction dynamique d'applications dans un environnement ouvert et hétérogène. En outre, ce paradigme fournit la dynamicité nécessaire à ces applications grâce à la gestion de l'apparition et la disparition des dispositifs physiques par des mécanismes d'annonces et de communication événementielles.

Notre thèse s'inscrit dans ce contexte et vise à répondre aux défis des systèmes AmI dans le cadre des architectures orientées services (SOA). Afin d'appliquer correctement ces architectures dans le cadre des systèmes intelligents ambiants, nous avons traduit les défis soulevés par ces systèmes à des modèles orientés service. C'est au cœur de ces modèles que se situe la composition de services. En effet, cette dernière est responsable de la construction dynamique d'applications en tenant compte, d'une part, des fonctionnalités offertes par les services disponibles et d'autre part, des besoins variables exprimés par les utilisateurs. Par conséquent, nous allons présenter plus en détail cette problématique de composition de services dans le prochain chapitre.

CHAPITRE 3

Composition de services

3.1. Introduction

Á l'inverse de l'informatique traditionnelle qui est orientée traitement, l'informatique ambiante s'intéresse à l'adaptation des services aux besoins de l'utilisateur. Comme nous l'avons mentionné dans le chapitre précédent, un service est indépendant de toutes plateformes logicielles ou matérielles et il peut être invoqué par des clients ou par d'autres services. Dans l'approche SOA, un service est vu comme une brique fondamentale dans le processus de construction d'applications. Cette construction est réalisée par la composition rapide des services disponibles dans l'environnement. En particulier, la composition de services prend tout son intérêt dans les environnements ambiants où plusieurs équipements offrent différents services pour satisfaire la demande des clients. Ces derniers peuvent êtres soit des utilisateurs simples ou bien d'autres applications logicielles. Lorsque les services fournis ne peuvent pas satisfaire individuellement la demande du client, il est alors possible, grâce à la composition de services, de les combiner ensemble afin de satisfaire cette demande.

Nous distinguons deux niveaux d'abstraction des services. Les services de base qui fournissent des fonctionnalités élémentaires, et les services composites qui agrègent un ensemble de fonctionnalités dans des applications de haut niveau proches des besoins de l'utilisateur. Les services composites sont construits par le processus de composition de services qui permet la liaison entre les besoins de l'utilisateur et les services de base disponibles. Les développeurs peuvent ainsi résoudre des problèmes complexes en combinant les services de base disponibles et les ordonner pour mieux satisfaire les exigences de leurs problèmes. De plus, le processus de composition de services accélère le développement d'applications, la réutilisation de services, et la réalisation de services plus complexes **[94].**

Dans ce chapitre, nous présentons tout d'abord quelques définitions et concepts liés à la composition de services. Nous présentons également une classification des différentes catégories de composition de services rencontrées dans la littérature. Ensuite, nous étudions les approches de composition de services et un ensemble de travaux proposés pour chaque approche. Enfin, nous détaillons les mécanismes d'adaptation des services au contexte.

3.2. Définitions et étapes de la composition de services

3.2.1. Définition de la composition de services

Dans le chapitre précédent, nous avons étudié la description, la publication, et la recherche de services élémentaires. Si l'objectif du concepteur d'une application n'est pas atteint par l'invocation d'un simple service élémentaire, alors le concepteur doit combiner les fonctionnalités d'un ensemble de services. Ce processus est appelé *composition de services* **[95, 96]**. Dans la littérature, plusieurs définitions ont été attribuées à la composition de services.

Nous retenons certaines d'entres elles qui nous paraissent les plus pertinentes. La composition de services a été définie par **[97]** comme étant la capacité d'offrir des services à valeur ajoutée en combinant des services existants probablement offerts par différentes organisations. Elle a été définie également par **[98]** comme étant une technique permettant d'assembler des services pour réaliser un objectif particulier, par l'intermédiaire de primitives de contrôles (boucles, tests, traitement d'exception, etc.) et d'échanges (envoi et réception de messages). Autrement dit, la composition de services spécifie quels services ont besoin d'être invoqués, dans quel ordre, quelles sont les données à échanger, et comment traiter les situations d'exceptions. La composition de services peut être vue comme un mécanisme qui permet l'intégration des services dans une application **[95]**. La définition la plus générale et la plus référencée dans la littérature est celle énoncée par **[95]**. Les auteurs considèrent la composition des services Web comme étant un moyen efficace pour créer, exécuter, et maintenir des services qui dépendent d'autres services. Nous constatons que les différentes définitions s'accordent sur le faite que la composition de services vise la création de nouveaux services, offrant de nouvelles fonctionnalités, à partir des services existants. Le nouveau service résultat d'une composition de services est appelé *service composite*. Son exécution nécessite alors l'invocation de plusieurs autres services afin de faire appel à leurs fonctionnalités. Les services invoqués lors d'une composition de services sont appelés *services composants*. La **figure 3.1** présente le principe de la composition de services; à partir d'un ensemble de services disponibles dans un registre, nous pouvons construire une application à services **[99].**

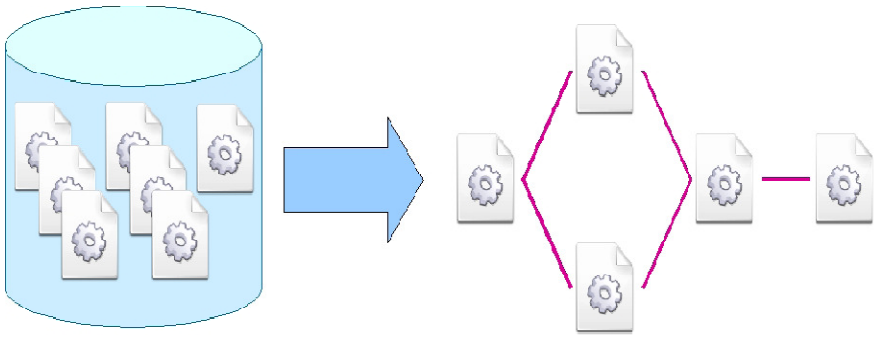


Figure 3.1 : Composition de services.

3.2.2. Etapes de la composition de services

La réalisation d'une application par composition de services comporte plusieurs étapes qui permettent le passage incrémental d'une spécification abstraite vers une composition concrète de services, c'est à dire une composition prête à être exécutée **[100, 68]**:

1. **Spécifications abstraites :** lors de cette phase, les fournisseurs de services décrivent et annonce leur services dans des annuaires publiques tels qu'UDDI. Cette description contient notamment les attributs fonctionnels et non-fonctionnels des services. Les attributs fonctionnels tels que les entrées et sorties sont nécessaires lors de la combinaison et l'invocation des services tandis que les attributs non-fonctionnels tels que le coût du service sont utilisés pour évaluer les services. Cette phase consiste également à identifier les fonctionnalités qui doivent être remplies par l'application résultant de la composition de services. Ces fonctionnalités peuvent êtres spécifiées sous forme d'une requête qui exprime les besoins de l'utilisateur. Cette requête peut être exprimée dans un langage

différent que celui utilisé pour la spécification des services. Dans ce cas, des translations sont effectuées entre le langage des requêtes (langage externe) et le langage des services (langage interne) afin d'assurer la compatibilité entre les besoins et les services.

2. **Génération des services composites:** cette phase consiste à identifier les services nécessaires à la composition afin de répondre aux besoins fonctionnels identifiés au préalable. Le générateur de composition (ou planificateur) tente de répondre à ces besoins en combinant, selon une stratégie donnée, les services disponibles. Le générateur prend souvent les fonctionnalités des services disponibles comme entrées, et éventuellement un modèle qui décrit le service composite. Ce modèle est un *Template* qui décrit le flot de contrôle et de données entres les services. Les services composites ainsi générés représentent des candidats pour la réalisation de la composition concrète de services.
3. **Evaluation des services composites:** il arrive souvent que plusieurs services possèdent des fonctionnalités similaires. Il est donc possible que le planificateur génère plusieurs plans de composition (services composites) qui satisfassent les besoins fonctionnels de l'application. Cette phase sélectionne alors, selon une stratégie donnée, un service composite parmi les candidats précédemment identifiés. La méthode la plus communément utilisée est les fonctions d'utilités. Dans ce cas, les services composites sont évalués en se basant sur les informations fournies par les attributs non fonctionnels des services composants. Les services sélectionnés sont préparés pour l'exécution: les services sont configurés et d'éventuelles adaptations sont réalisées.
4. **Déploiement et exécution des services:** lors de cette phase, les services sélectionnés et préparés au préalable sont déployés sur les plates-formes d'exécution. Il est ainsi possible d'invoquer ces services pour réaliser correctement et concrètement leur composition. L'exécution d'un service composite peut être vue comme une séquence de messages échangés selon le modèle de composition. Le flot de données du service composite est défini par le transfère des données de sortie d'un service composant aux prochains services composants qu'il précède directement dans le service composite.

Chaque étape peut être décomposée en plusieurs tâches que les développeurs exécutent parfois manuellement. Les développeurs doivent résoudre les différentes incompatibilités des services participants à la réalisation d'une composition de services, comme par exemple celle des types de données d'entrée et de sortie. La complexité de la composition est double **[99]**: d'une part, la complexité de la logique métier de l'application qui nécessite une expertise métier dans le domaine visé; d'autre part, la complexité technique de la réalisation de l'approche à services notamment la description, la recherche, la sélection et l'invocation des services. Un aspect donc essentiel pour simplifier le travail des développeurs consiste à automatiser une partie des tâches associées à la composition de services.

3.3. Catégories de composition de services

Ces dernières années, plusieurs communautés, aussi bien dans la recherche académique que dans l'industrie, se sont intéressées à la problématique de la composition de services. Cette problématique est à l'origine d'un nombre considérable de travaux notamment dans les communautés des bases de données **[101, 102]**, du web sémantique **[103, 85, 104]**, et plus récemment dans celles de l'intelligence artificielle **[105, 106, 107]**. La **figure 3.2** présente

notre classification des catégories de composition de services proposées et rencontrées dans la littérature. Cette classification se base sur trois axes principaux:

1. **Degré d'automatisation** : en fonction du degré de participation de l'utilisateur (*user involvement*) dans la définition du schéma de composition, trois types de composition de services sont distingués : compositions manuelles, semi-automatiques ou automatiques.
2. **Moment de la sélection de services** : la réalisation concrète d'un schéma de composition de services nécessite le passage d'une spécification abstraite de la fonctionnalité attendue à une application exécutable. Ce passage est réalisé par la sélection de services qui consiste à identifier les services concrets à invoqués lors de l'exécution. Selon que cette sélection soit faite a priori ou non, c'est-à-dire, aux moments de la conception (*design time*) et du déploiement ou bien au moment de l'exécution, une composition est dite statique ou dynamique.
3. **Contrôle de l'exécution de la composition de services**: selon que la responsabilité de la coordination de l'exécution de la composition de services soit confiée à une seule entité ou bien partagée entre les différents fournisseurs des services, une composition est dite centralisée ou distribuée.

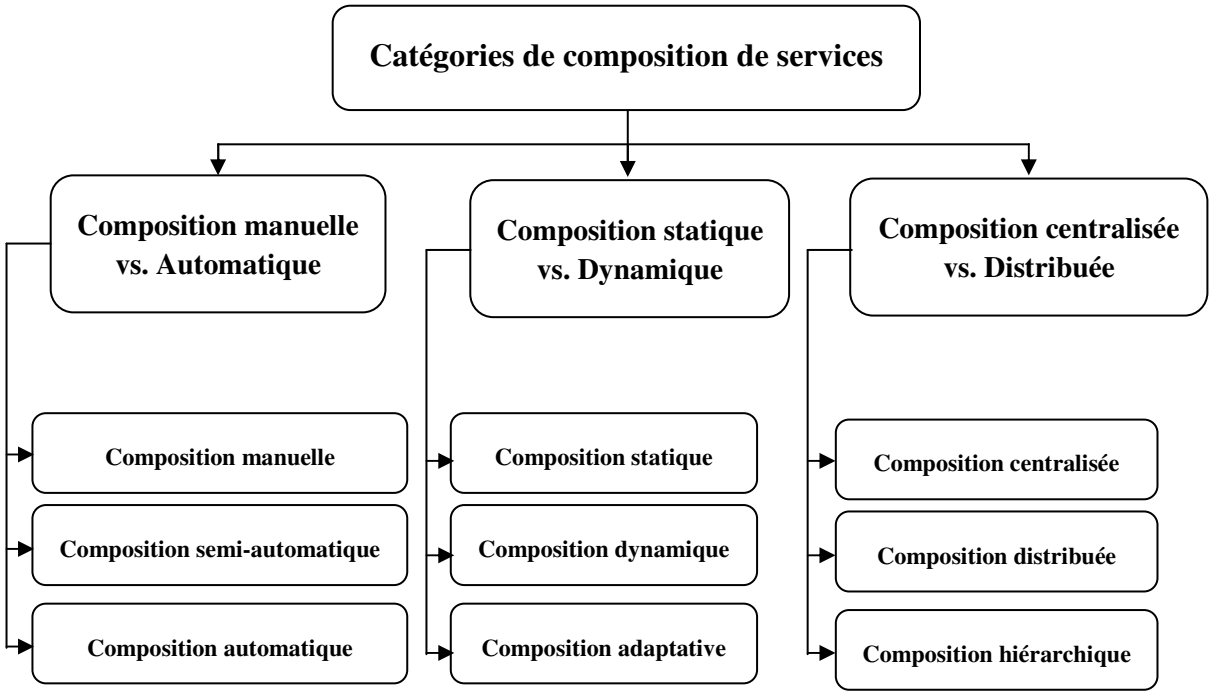


Figure 3.2 : Catégories de composition de services.

3.3.1. Composition manuelle vs. Automatique

3.3.1.1. Composition manuelle

Les techniques de composition manuelles reposent sur un expert en charge de définir et générer, graphiquement ou moyennant un éditeur de texte, le schéma de la composition de services. Dans cette composition manuelle, l'utilisateur joue un rôle central en définissant manuellement l'ordre d'invocation et d'exécution des services composants. Pour ce faire, l'utilisateur se charge d'abord de définir sa requête qui reflète son besoin en termes de composition de services. Ensuite, il définit le schéma de composition de services qui répond à

sa requête en spécifiant les flux de données et de contrôle. Dans cette étape, l'utilisateur se base essentiellement sur sa connaissance sur le domaine (expertise) et la consultation du registre de services disponibles. Enfin, le schéma de composition de services défini est soumis à un moteur d'exécution afin de le réaliser concrètement.

3.3.1.2. Composition semi-automatique

Dans les approches semi-automatiques, des outils graphiques sont développés afin de guider l'utilisateur étape par étape durant le processus de composition. Dans une première étape, ces outils mettent à disposition un certain nombre d'opérateurs graphiques pour aider l'utilisateur voulant construire un service composite sous forme d'un workflow. Dans cette étape, l'outil peut exploiter les connaissances du domaine pour mieux guider l'utilisateur dans sa construction du workflow. Dans une seconde étape, l'outil propose à l'utilisateur des conseils et des suggestions quant à la sélection des services à composer (ex. en fonction de leurs entrées/sorties, annotations, etc.). Une fois le service composite est défini, l'outil instancie les opérateurs de contrôle ainsi que le workflow correspondant à ce service composite. Le workflow instancié est enfin soumis à un moteur d'exécution. Certains outils permettent également de mémoriser les workflows pour faciliter leurs réutilisations.

3.3.1.3. Composition automatique

Dans ce type de composition, l'utilisateur ne fait que spécifier ses besoins sous forme d'une requête de services. Par la suite, c'est le système qui prend en charge tout le processus de composition et le réalise automatiquement, sans qu'aucune autre intervention de l'utilisateur ne soit requise. Le système automatise entièrement le processus de composition en le traitant généralement comme un problème de planification. La tâche de composition consiste alors à trouver un ou plusieurs plans satisfaisant un but de composition dans le domaine des services disponibles. Pour ce faire, l'algorithme de planification se base sur des descriptions compréhensibles des services afin de trouver les services dont les fonctionnalités conviennent aux besoins de l'utilisateur. Les plans générés par cet algorithme de planification sont des chaînes de services (services composites) pouvant être traduits en code exécutable pour être exécutés dans l'ordre. Ainsi, la tâche de composition est réalisée de manière tout à fait transparente à l'utilisateur.

3.3.2. Composition statique vs. Dynamique

3.3.2.1. Composition statique

La composition statique est définie dans **[108]** comme suit : « On dit qu'une *composition* est *statique* lorsqu'elle prend place à l'étape de conception, au moment où l'architecture et la conception du système logiciel sont planifiées. Les composants (ou services) qui seront utilisés sont préalablement choisis et reliés, et la gestion du flot est effectuée a priori ». Ainsi, une composition statique est une composition qui est spécifiée aux moments de la conception et de déploiement de l'architecture du système. Les services à composer sont identifiés, sélectionnés, interconnectés, compilés et déployés par le concepteur. Cela donne lieu à des compositions figées, avec des services participants et des liaisons entre eux prédéfinis. Dans ce cas de figure, les concepteurs ne contrôlent pas le cycle de vie des services qu'ils conçoivent et utilisent. Ce type de composition n'est rentable que lorsque le système et les

services sont peu évolutifs. Les approches statiques de composition de services sont en général adoptées par l'industrie. Elles s'inspirent de la gestion de processus métiers quant à la description des données et des flots de contrôle pour le processus de composition **[109]**.

3.3.2.2. Composition dynamique

La composition dynamique est définie dans **[110]** comme suit : « une composition de services est dite *dynamique* si les services sont sélectionnés et composés à la volée en fonction des besoins formulés par l'utilisateur ». Par opposition à la composition statique, la composition dynamique se caractérise par le faite que les services candidats sont localisés par le système à l'arrivé de la requête, puis reliés à la demande de l'utilisateur pour obtenir un service composite. Ainsi, la sélection des services participants et la réalisation des liaisons entre eux sont retardées jusqu'au moment de l'exécution. Les étapes de planification et de construction du service composite sont réalisées au moment de l'exécution en fonction des fournisseurs de services disponibles à ce moment. Dans un premier temps, ce type de composition peut engendrer de nombreuses applications utiles qui n'ont pas été prévues à l'étape de conception. Dans un second temps, ces applications sont réalisées de manière flexible et adaptable en sélectionnant et en combinant les services appropriés sur la base de la requête et du contexte de l'utilisateur. Par conséquent, la composition dynamique de services est propice pour répondre aux exigences des environnements dynamiques où les composants disponibles changent constamment et les attentes des utilisateurs sont variables et personnalisées. Cependant sa mise en œuvre reste difficile à cause des changements fréquents de contextes des utilisateurs et des services dans de tels environnements.

3.3.2.3. Composition adaptative

Dans la composition statique, les liaisons entre l'application cliente et les services utilisés sont définies avant l'exécution (liaisons statiques), tandis que, dans la composition dynamique, ces liaisons sont réalisées à l'exécution (liaisons dynamiques). L'hybridation de ces deux types de liaisons a engendré un troisième type de liaison appelé liaison adaptative. Dans ce cas, la composition est réalisée à manière d'une composition statique, avec des liaisons décidées avant l'exécution, mais qui peuvent être changées si besoin en fonction de certaines stratégies. Si un fournisseur de services n'est plus disponible au moment de la réalisation de la liaison avec le service fourni, la composition adaptative peut choisir un fournisseur compatible et se lier au service qu'il met à disposition **[111].**

3.3.3. Composition centralisée vs. Distribuée

3.3.3.1. Composition centralisée

Cette catégorie nécessite un nœud central qui agit comme un coordinateur de composition de services et imposant de ce fait une structure centralisée. Ce nœud central, appelé aussi coordinateur central ou moteur d'exécution, prend le contrôle de tous les services intervenant dans une composition de services. Son rôle principal consiste à gérer l'invocation des différents services impliqués dans le schéma de composition de services.

3.3.3.2. Composition distribuée

Contrairement à la composition centralisée, la composition distribuée n'a pas de coordinateur central. Ce type de composition impose une structure décentralisée dans laquelle, tous les

services participants à une composition de services doivent collaborer afin de réaliser cette composition. C'est ainsi que la tâche de réalisation d'un service composite est, cette fois-ci, répartie entre tous les services concernés.

3.3.3.3. Composition hiérarchique

A défaut de considérer une structure purement centralisée ou purement distribuée, la composition hiérarchique combine ces deux structures. Dans ce cas de figure, la topologie du réseau est partitionnée en plusieurs sous-ensembles. Chacun de ces ensembles possède son propre coordinateur local de composition de services. Les différents coordinateurs locaux sont, à leurs tours, rattachés à un autre coordinateur de niveau hiérarchique supérieur. Le nombre de niveaux hiérarchiques dépend de l'étendu du réseau et du nombre de services.

3.4. Approches de composition de services

D'un point de vu technique, les différentes catégories illustrées précédemment peuvent être réalisées par plusieurs approches de composition de services. A la lumière de notre étude des différents travaux sur la composition de services, nous avons identifié cinq approches principales: la composition par procédés, la composition structurelle, la composition par planification, la composition sémantique, et la composition par intergiciels. Chaque approche fourni un ensemble de techniques afin de réaliser concrètement une composition de services. La **figure 3.3** illustre notre classification de ces approches de composition de services.

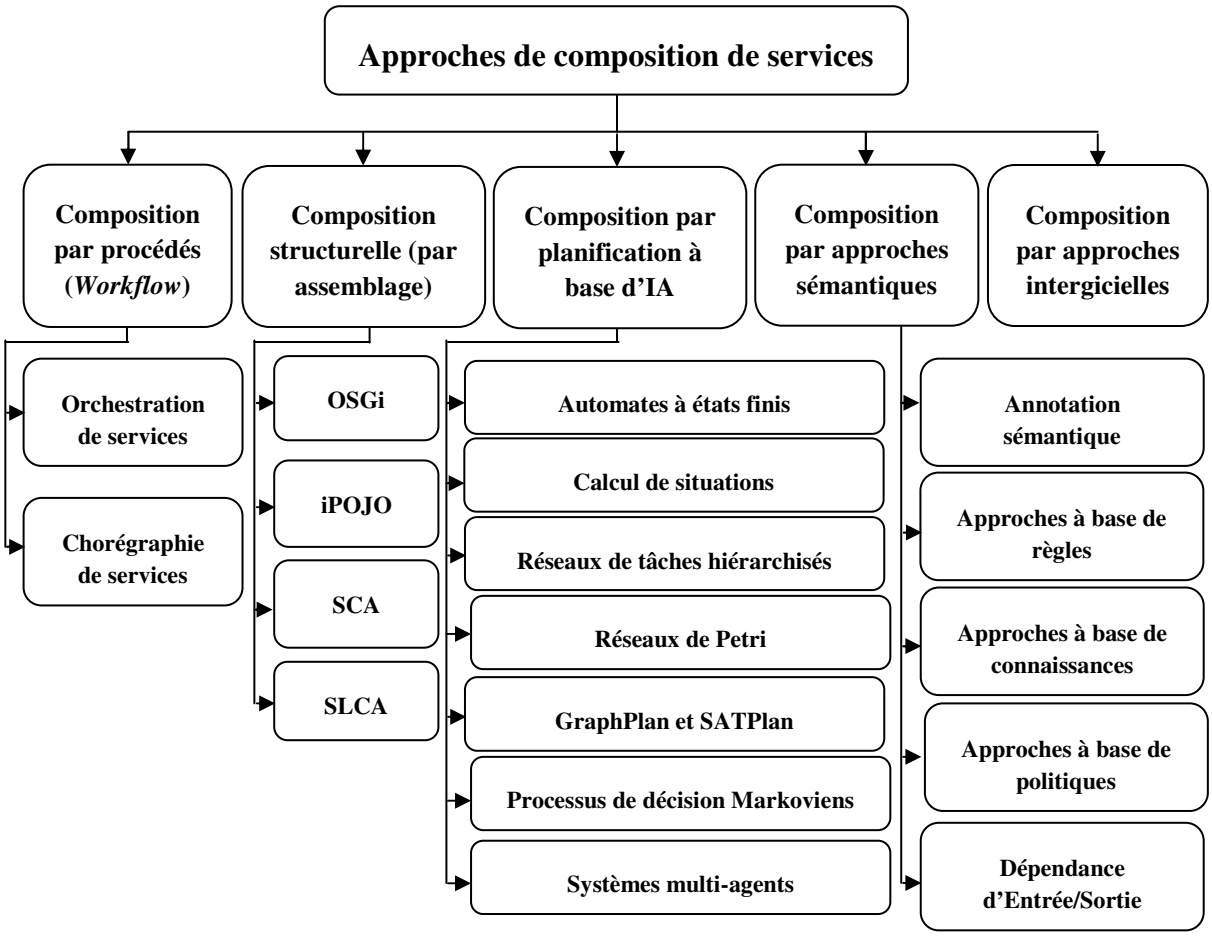


Figure 3.3 : Approches de composition de services.

3.4.1. Composition par procédés (*Workflow*)

Dans cette approche, le schéma de la composition de services est défini en spécifiant la logique de coordination des services par un procédé. Un procédé, également connu sous le nom du *Workflow*, est représenté par un graphe orienté d'activités et un flot de contrôle qui donne l'ordre d'exécution des activités (**Figure 3.4**). Chaque activité représente une fonctionnalité fournie et réalisée par un service. Cette approche de composition se base ainsi sur la grande similitude qui existe entre un service composite et un Workflow. Un service composite comporte un ensemble de services atomiques ainsi que les contrôles et les échanges de données entre ces services. De la même façon, un Workflow est composé d'un ensemble d'activités élémentaires structurées, ainsi que l'ordre d'exécution entre elles. En faisant l'analogie entre les activités et les services atomiques, un service composite est alors vu comme un Workflow dans lequel, chaque activité composante correspond à l'invocation d'un service et le flot de contrôle définit les conditions et l'ordre d'invocation des services participants et les différentes interactions entre ces services. En pratique, un Workflow est décrit dans un langage spécifique qui est interprété par un moteur d'exécution. Ce dernier se charge de réaliser l'invocation des services correspondants aux activités du Workflow, la communication entre les services impliqués (routage des événements et des messages d'un service à un autre) et la gestion des erreurs.

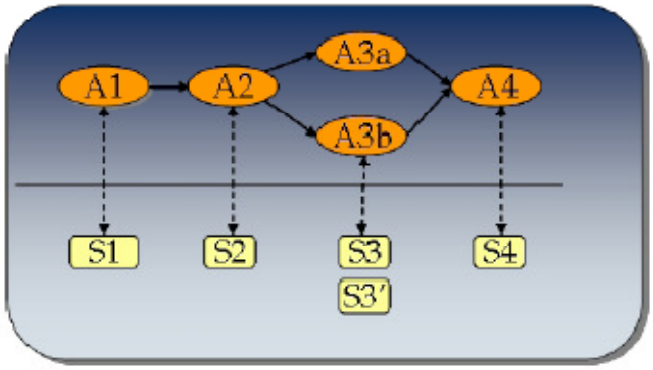


Figure 3.4 : Composition par procédés.

La composition par procédés permet de construire une application (service composite) à base de briques logicielles (services) dont le fonctionnement interne n'est pas connu « boîtes noires». La seule chose qui reste connue de l'application, c'est le flot de contrôle de l'agencement de ses services composants. Ce mode de contrôle explicite permet ainsi un contrôle externe des invocations des services constituants l'application sans se soucier de leurs fonctionnements internes. Actuellement, ce type de composition de services est utilisé pour les services Web dans lesquelles le consommateur des services ne détient pas de contrôle des fournisseurs des services impliqués. En fonction du type de contrôle de la composition, deux catégories de compositions de services par procédés ont été identifiées : l'orchestration de services et la chorégraphie de services. Ces deux points de vue de la composition par procédés ne sont utilisés actuellement que pour la technologie des services Web **[99].**

3.4.1.1. Orchestration de services

L'orchestration décrit un ensemble d'actions dans lesquelles un service donné peut ou doit s'engager. Elle offre une vision centralisée de la logique de coordination d'une composition de services. Un coordinateur central - appelée moteur d'exécution- prend le contrôle de tous les services intervenant dans une composition de services. Il gère l'invocation de ces différents services selon la logique définie par le procédé. Les auteurs dans **[112]** définissent l'orchestration comme un ensemble de processus exécutés dans un ordre prédéfini afin de répondre à un but. D'après **[113]**, l'orchestration de services Web consiste en la programmation d'un moteur qui appelle un ensemble de services Web selon un processus prédéfini. Ce moteur définit le processus dans son ensemble et appelle les services Web (tant internes qu'externes à l'organisation) selon l'ordre des tâches d'exécution. Les services Web impliqués dans une composition n'ont pas de connaissance d´être partie d'une composition. Seulement le coordinateur de l'orchestration a besoin de cette connaissance. La **figure 3.5** illustre l'orchestration de deux services Web (*Web Service 1* puis *Web Service 2*) réalisée par le coordinateur (*Web Service Coordinator*).

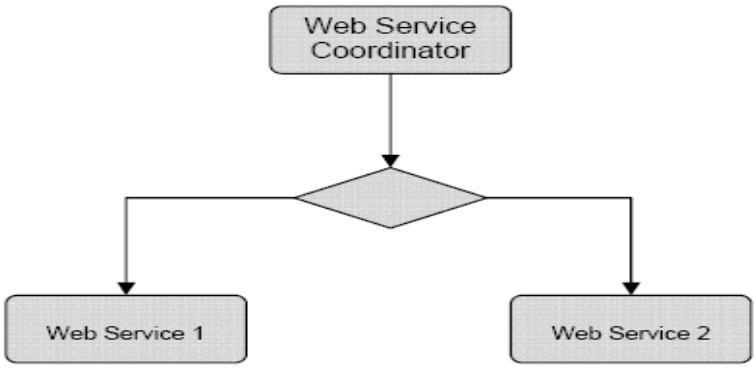


Figure 3.5 : Orchestration de services.

Il existe plusieurs langages d'orchestration pour les services Web. BPEL (*Business Process Execution Language*) **[114]** est un exemple de langage d'orchestration de services Web qui est le plus utilisé actuellement.

3.4.1.2. Chorégraphie de services

Contrairement à l'orchestration, la chorégraphie n'a pas de coordinateur central. Elle s'intéresse plutôt à la manière dont les services participants doivent collaborer afin de réaliser un but commun. La logique de la collaboration est, cette fois-ci, répartie entre les services concernés. Comme indiqué dans **[112]**, la chorégraphie permet de décrire la composition comme un moyen d'atteindre un but commun en utilisant un ensemble de services Web. La chorégraphie est ainsi un effort de collaboration dans lequel chaque service participant connaît exactement quand ses opérations doivent être exécutées et avec qui l'interaction doit avoir lieu. D'après **[111],** chaque participant impliqué dans une chorégraphie joue un rôle attribué. La chorégraphie décrit d'une manière globale les échanges de messages (les conversations), les règles auxquelles sont soumises les interactions des différentes parties, respectivement la manière dont les différents services se coordonnent afin de remplir le but de la composition de services. La **figure 3.6** représente une collaboration entre trois services Web dans le cadre d'une chorégraphie. Plusieurs propositions de langages pour la

modélisation de la chorégraphie de services existent : WS-CDL (*Web Services Choreography Description Language*) **[115]**, WSCI (*Web Services Collaboration Interface*) **[116]** et ebXML (*Electronic Business using eXtensible Markup Language*).

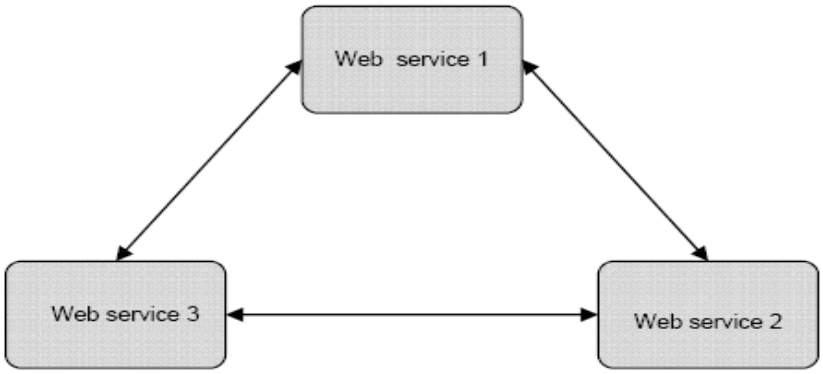


Figure 3.6 : Chorégraphie de services.

L'orchestration est un rapprochement plus flexible que la chorégraphie pour le processus de composition de services. En effet, le coordinateur de tout le processus est connu, et les services Web peuvent être insérés sans soucis, parce qu'ils n'ont pas connaissance d'appartenir au processus. Toutefois, l'approche par chorégraphie est plus adaptée si les services sont vus comme des agents qui interagissent entre eux afin de mener le processus de composition. Par ailleurs, l'orchestration et la chorégraphie peuvent être utilisées conjointement pour permettre l'intégration des systèmes des différentes entreprises **[117]**. Ainsi, la chorégraphie modélise la collaboration entre les différentes partenaires impliqués et l'orchestration est utilisée en interne par chaque partenaire pour modéliser la réalisation de la fonctionnalité rendue disponible par ceux-ci.

3.4.2. Composition structurelle (par assemblage)

Dans ce modèle de composition, les services sont fournis par des composants. Chaque composant spécifie clairement, dans des interfaces, les services fournis ainsi que les services requis (dépendances de services) qui lui sont nécessaires afin de réaliser sa logique métier. La composition structurelle permet ainsi de décrire une composition de services comme un assemblage de composants.

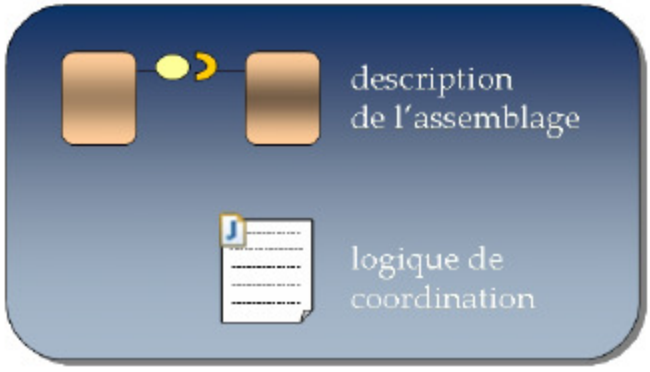


Figure 3.7 : Composition structurelle.

L'assemblage de composants consiste à configurer et lier les composants qui fournissent les services nécessaires à la composition. En d'autres termes, il s'agit de chercher les correspondances entre les services fournis et les services requis par les composants. Cette correspondance se base notamment sur les aspects syntaxiques et sémantiques afin d'assurer une composition valide. Par conséquent, dans une composition structurelle, les services à composer sont clairement identifiés avec leurs interactions lors de la phase d'assemblage de composants. La logique de coordination entre ces services, qui définit la manière dont la composition de services fonctionne, est spécifiée par le développeur et livrée avec l'assemblage. Par exemple, la **figure 3.7** montre un assemblage de deux composants à services dont la logique de coordination est réalisée par une classe Java [**111**].

Ainsi, par opposition à la composition par procédés, le contrôle dans une composition structurelle est interne (à l'intérieur des services) et connu seulement par son développeur. En effet, l'assemblage est réalisé au niveau des implémentations des services (composants) en utilisant les mécanismes spécifiques définis par le modèle à composants orientés service respectif. Dans ce modèle, les services sont implantés sous forme de composants. Plusieurs modèles à composants pour la composition de services ont été proposés **[118, 94]**. Dans cette section, nous présenterons le modèle SCA (*Service Component Architecture*) pour la composition structurelle de services Web et le modèle iPOJO (*injected POJO*) pour la composition structurelle de services OSGi.

3.4.2.1. OSGi

OSGi **[119]** est une spécification de plate-forme à services qui a été proposée, au début des années 2000, par l'Alliance OSGi, un consortium composé de nombreux grands acteurs industriels comme IBM, Motorola, Nokia et Alcatel. Actuellement, il existe plusieurs implantations de la plate-forme OSGi telles que Felix de la communauté Apache et Equinox d'IBM. La plate-forme OSGi à été définie au dessus de la plate-forme Java. Elle offre un environnement de déploiement et d'exécution orienté services, un modèle d'implémentation orienté composants et un certain nombre de services techniques **[111]**. Dans cette plate-forme, les services sont implémentés et déployés sous forme de composants appelés « bundles ».

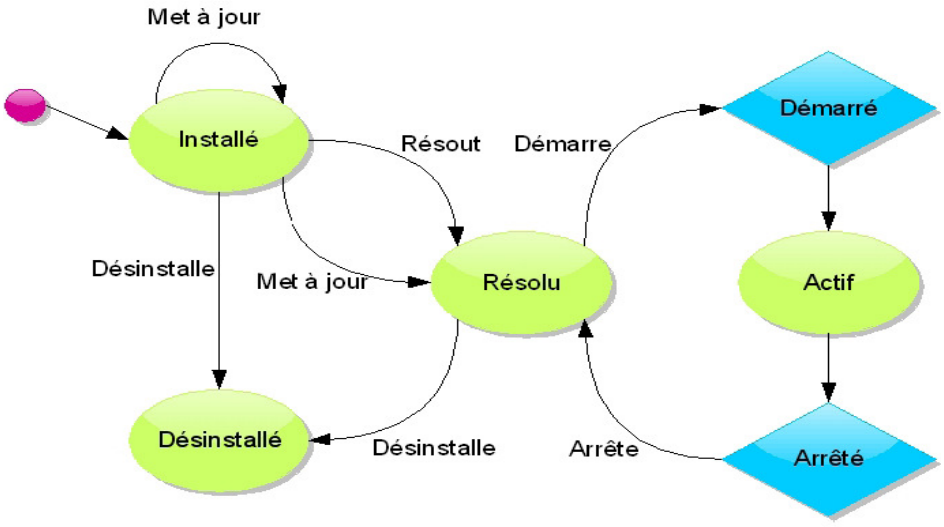


Figure 3.8 : Cycle de vie d'un *bundle* OSGi **[99]**

Physiquement, un bundle est une unité de déploiement qui correspond à un fichier JAR augmenté par des métadonnées. Ces dernières spécifient essentiellement les dépendances de code du bundle avec les autres unités de déploiement (des packages importés) et les fonctionnalités fournies par le bundle (des packages exportés). Cette caractéristique permet de contrôler au démarrage d'un bundle que toutes les dépendances de code sont résolues. La plate-forme OSGi, qui représente l'environnement d'exécution des bundles, fournit un ensemble de mécanismes pour gérer leur cycle de vie comme le montre la **figure 3.8**. Cette plate-forme permet le déploiement et l'administration des bundles à l'exécution. En effet, elle permet l'installation, le démarrage, l'arrêt ou encore la mise à jour des bundles à l'exécution sans interruption de la plate-forme. Cependant, la gestion des dépendances et des interactions entres les bundles est effectuée manuellement par le développeur. Cette tâche reste complexe car elle demande une grande connaissance des mécanismes OSGi. Afin de remédier à ce problème, la technologie iPOJO propose de gérer automatiquement ces interactions.

3.4.2.2. iPOJO

iPOJO (*injected Plain Old Java Object*) est un modèle à composants orientés service développé au dessus d'OSGi (comme sous-projet du projet Apache Felix) au sein de l'équipe Adèle du Laboratoire d'Informatique de Grenoble (LIG). L'objectif principal d'iPOJO est de simplifier le développement des compositions structurelles de services **[120, 121]** grâce notamment à la réalisation des liaisons dynamiques de services. En effet, un composant iPOJO expose des fonctionnalités à travers des services et exprime des dépendances vers d'autres services comme le montre la **figure 3.9**. Ces dépendances sont exprimées comme des services abstraits, fait qui permet de retarder la réalisation des liaisons jusqu'à l'exécution. La réalisation des liaisons dynamiques est déléguée au containeur du composant iPOJO. Un containeur est composé de plusieurs handlers, chacun d'entre eux prend en charge un aspect non-fonctionnel telles que la résolution des dépendances de services, la publication des services fournis, la configuration des composants, etc.

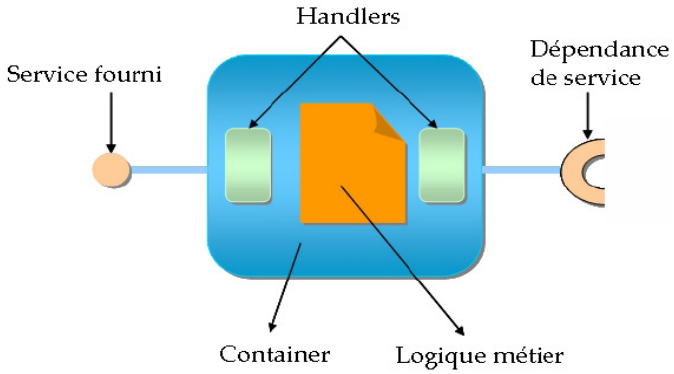


Figure 3.9 : Les éléments d'un composant iPOJO **[111]**

Grâce à ce mécanisme de containeur, iPOJO permet de simplifier énormément la tâche des développeurs des composants iPOJO. En effet, le développeur doit seulement implémenter la logique métier de la fonctionnalité publiée par son composant et configurer le containeur de son composant. Par la suite, le containeur gère automatiquement les tâches qui lui sont déléguées en utilisant l'information de configuration exprimée par le développeur.

3.4.2.3. SCA

SCA (*Service Component Architecture*) est un modèle à composants orienté services spécifié par le consortium OSOA **[122]**. Un composant SCA implémente un ou plusieurs services pour réaliser une certaine logique métier. Plusieurs technologies peuvent être utilisées pour réaliser cette implémentation, comme par exemple, Java, PHP, C++, etc. Cet aspect est considéré, dans SCA, comme étant un aspect purement technique, donc secondaire **[123]**. Comme le montre la **figure 3.10**, un composant SCA est constitué de trois parties : les services fournis, les références de services et les propriétés associées.

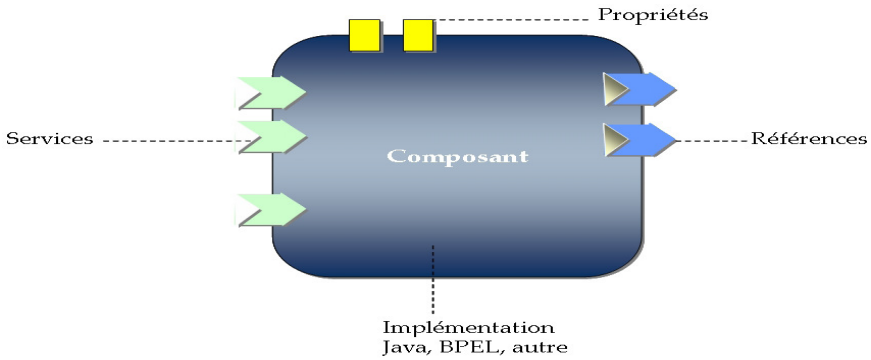


Figure 3.10 : Vue externe d'un Composant SCA **[111]**

Les services fournis réalisent l'implémentation interne de la logique métier du composant à travers un ensemble d'opérations. Ces services sont exposés et décrit à travers des interfaces de services. Les références de services décrivent les dépendances de services. Ce sont des services externes utilisés pour l'implémentation interne des services fournis. Chaque référence définit une interface contenant les opérations que le composant utilise. Un composant peut également déclarer une ou plusieurs propriétés qui seront utilisées lors de sa configuration. La configuration d'un composant consiste à spécifier les valeurs pour les différentes propriétés qu'il expose et la manière dont les liaisons entre le monde extérieur et le composant seront réalisées, par l'intermédiaire des références de services [**124**]. La réalisation effective des liaisons aux services référencés est effectuée par injection des dépendances **[125]** lors de l'exécution. Cela élimine une partie de la complexité technique de la résolution des dépendances pour le développeur **[111]**. Dans le modèle SCA, la composition est réalisée par des assemblages de composants SCA. Ces assemblages, appelés composites **[126]**, contiennent des composants et les liaisons qui décrivent les connections entre composants. Cependant, cet assemblage est réalisé au niveau conceptuel et non au niveau exécution, et le modèle SCA ne fait aucune proposition quant à l'adaptation dynamique d'assemblages à l'exécution.

3.4.2.4. SLCA

SLCA (*Service Lightweight Component Architecture*) **[127]** est un modèle d'architecture, inspiré du modèle SCA, pour la composition de services à base d'assemblage de composants légers. SLCA est un modèle multi-paradigmes utilisant : services, composants légers et événements. Le middleware WComp **[128]** en est l'implémentation de référence. Afin de réaliser des compositions de services, à base d'événements et d'assemblage de composants légers, des composants proxy vers des services Web ou des services pour dispositifs sont

générés dynamiquement et intégrés dans un assemblage. Un assemblage de composant se trouve dans un conteneur qui peut lui-même être exporté comme un service composite. Le modèle est donc hiérarchique puisque ce service pourra être utilisé dans un autre service composite. Les communications entre les composants sont événementielles. Le modèle SLCA s'appuie sur un environnement d'exécution dynamique. Cet environnement est défini comme un ensemble de ressources, qui sont des entités logicielles ou matérielles fournis à l'application. Ces entités peuvent apparaître et disparaître durant l'exécution.

L'approche de composition SLCA est basée sur des composants légers **[129]** similaires à JavaBeans et OpenCom **[130]**. Un service composite encapsule le containeur SLCA qui contient un assemblage de composants dynamique et léger dérivés du modèle LCA (*Lightweight Component Architecture*). LCA est un modèle dérivé de Beans **[131]**, adapté aux autres langages de programmation, avec des concepts de ports d'entrée et de sortie et de propriétés. Ces composants sont appelés «*light*» pour plusieurs raisons **[128]**. La première est qu'ils s'exécutent dans le même espace d'adressage mémoire, et dans le même processus. Leurs interactions sont ainsi réduites à la plus simple et la plus efficace manière: l'appel de fonction. La deuxième raison, c'est qu'ils n'incorporent pas du code non fonctionnel ou d'autres services techniques non-obligatoires. Leur empreinte mémoire est alors réduite et ils sont instanciable et destructibles rapidement. La troisième raison est qu'ils ne contiennent pas de référence entre eux au moment de la conception (*design-time*), et respectent les concepts de boîtes noires et de liaisons tardives. Les composants légers sont totalement découplés, et très réactifs car ils utilisent des événements pour communiquer entre eux. Le diagramme UML du modèle LCA est décrit par la **figure 3.11**. Les composants ont une interface, définie par le type du composant. Cette interface est un ensemble de ports d'entrée (méthodes), et les ports de sortie (événements), chacun étant typé par ses paramètres, et ayant un identifiant unique. Les interactions entre les composants sont des liaisons ou des liens.

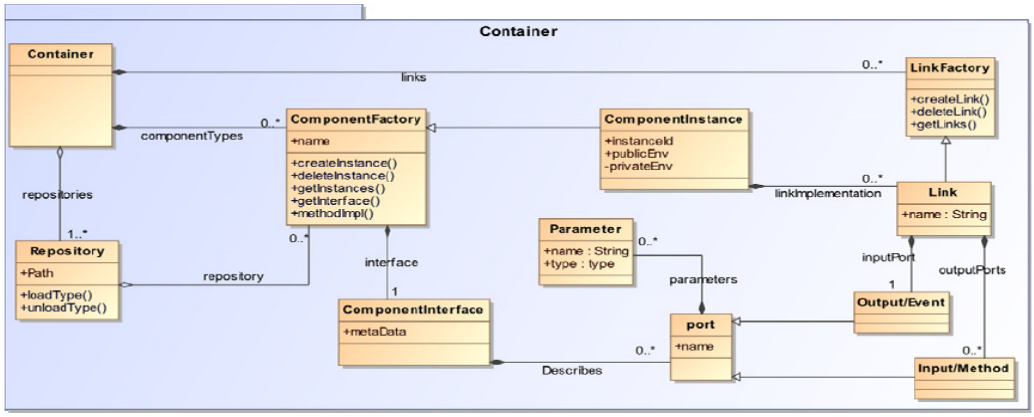


Figure 3.11 : Méta-modèle LCA: composants légers **[128]**

Le diagramme UML du modèle SLCA est décrit par la **figure 3.12.** Les *proxy components* permettent aux services de l'environnement d'être utilisés dans le service composite, tandis que les *probe components* permettent aux nouveaux services d'êtres ajouter à l'environnement et éventuellement d'êtres utilisés par d'autres services composites. Le concept de hiérarchie distribuée est ensuite introduit, à travers la couche service.

Ils existent deux types de composant *probe components*: *sinks* et *source. Sinks* ajoutent une méthode à l'interface de service composite. Dans son assemblage interne, le composant *sink* ne dispose que d'un port de sortie. L'invocation de la méthode de l'interface de service émet donc un événement dans l'assemblage du composant. Le second type est la *source* qui ajoute un événement à l'interface de service composite, et comporte seulement un port d'entrée. L'invocation de la méthode de l'interface de composant émet donc un événement de service Web. De plus, un service composite exporte deux interfaces : (1) l'interface structurelle, qui permet de gérer le contenu du composite (ajouter/retirer des composants/liaisons ou encore modifier des propriétés) et (2) l'interface fonctionnelle qui permet de communiquer avec les composants internes du composite.

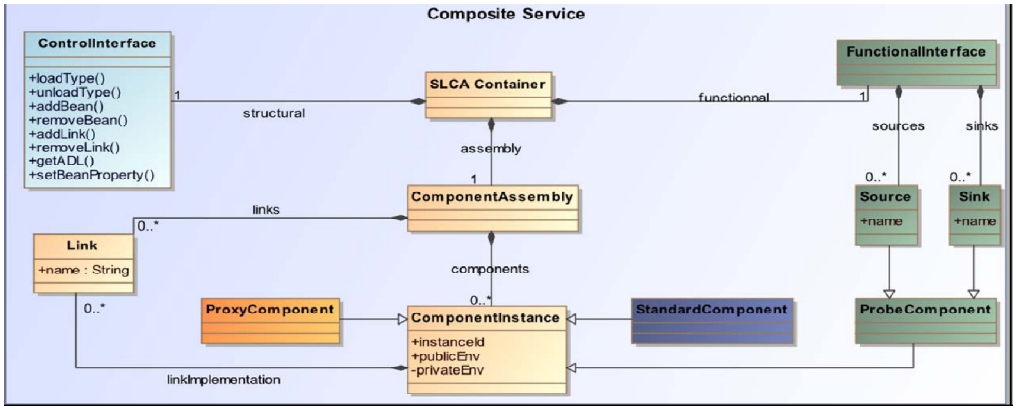


Figure 3.12 : Méta-modèle SLCA: interfaces des services composites **[128]**

3.4.3. Composition par planification à base d'IA

Dans cette approche, la composition de services est vue comme un problème de planification visant à atteindre un certain objectif. Ce problème a été étudié en détail par la recherche en intelligence artificielle (IA). Un problème de planification classique comprend une description de l'état initial du monde réel, une description de l'objectif souhaité, et une description des actions possibles qui peuvent être exécutées. Étant donné une représentation des services comme des actions, le problème de composition de services se ramène à un problème de planification de telle sorte qu'à partir de l'objectif de l'utilisateur et un ensemble de services, un planificateur doit trouver une collection de services qui aboutissent à cet objectif. Cette ressemblance démontre la pertinence de la planification à base de l'IA pour entreprendre le problème de composition de services.

L'hypothèse de base des méthodes de planification à base de l'IA est que chaque service est une action qui modifie l'état du monde réel à travers son exécution. Puisque les services (actions) sont des composants logiciels, leurs paramètres d'entrée/sortie agissent respectivement comme des préconditions et des effets dans le contexte de la planification. Si l'utilisateur peut spécifier les entrées et sorties requises par le service composite alors ce dernier peut être généré automatiquement par des planificateur sans aucune connaissance d'un workflow prédéfini. Dans cette section du document, nous donnons un aperçu des méthodes de planification à base d'IA qui ont été étudiées dans le cadre de la composition de services.

3.4.3.1. Automate à états finis

Un automate est un modèle très connu dans le domaine de la spécification formelle des systèmes complexes. Les automates à états finis (FSA – *Finite State Automata*) est une classe d'automates composée d'un ensemble fini de nœuds représentant les états, un ensemble d'actions et un ensemble d'arcs étiquetés décrivant les transitions entre les états. Cette classe d'automates est adoptée dans certains travaux pour l'étude des langages formels et la spécification des processus de composition de services. Dans **[132]**, les auteurs mettent l'accent sur les services Web dont le schéma (à la fois interne et externe) peut être représenté à l'aide d'un FSA et proposent un algorithme qui vérifie l'existence d'une composition de services. Un type spécifique de FSA, qui génère une sortie en fonction de son état actuel et d'une entrée, appelé machines de Mealy (*Mealy Machines*), a été proposé pour la modélisation des services **[133]**. Selon cette approche, les services communiquent en envoyant des messages asynchrones, et chaque service possède une file d'attente. Un «superviseur» global garde une trace de tous les messages échangés. La conversation entre services est alors introduite sous forme d'une séquence de messages. En étudiant et en comprenant les propriétés d'une conversation, l'approche prévoit de nouvelles méthodes pour la conception et l'analyse des compositions de services. Dans **[149]**, une approche permettant la description des processus OWL-S en automates à états finis a été proposée dont l'objectif est de sélectionner les meilleurs services composés en fonction des requêtes des utilisateurs exprimées aussi par des processus OWL-S.

3.4.3.2. Calcul de situations

En calcul de situations (*Situation calculus*), un monde dynamique est modélisé comme étant un monde progressant à travers une série de situations à la suite de diverses actions exécutées dans ce monde. Une situation représente une histoire d'occurrences d'actions. La constante s_0 décrit la situation initiale où aucune action n'a encore eu lieu. La transition vers la situation successeur d'une situation *s* comme résultat d'une action *a* est notée *do(a, s)*. Enfin, les déclarations décrivant une situation sont modélisées par des symboles de prédicats logiques. Dans **[134]**, les auteurs considèrent la composition de services comme une arborescence d'exécution des actions. En s'appuyant sur un raisonnement sur des formalismes d'actions, il sera alors possible d'utiliser l'approche par calcul de situations. Dans **[135]**, les auteurs suggèrent l'utilisation d'une version étendue de Golog. Ce dernier est un langage de programmation logique de haut niveau, construit au-dessus du modèle de calcul de situations, pour la spécification et l'exécution des actions complexes dans des domaines dynamiques. Les auteurs étendent Golog par des connaissances et des actions de détection afin de devenir un formalisme approprié pour représenter et raisonner sur le problème de composition de services Web sémantiques. L'idée générale de leur méthode consiste à doter des agents logiciels d'une capacité de raisonnement sur les services Web afin d'effectuer automatiquement la découverte, l'exécution et la composition de services.

3.4.3.3. Réseau de tâche hiérarchique

Le concept de réseau de tâche hiérarchique (HTN – *Hierarchical Task Network)* **[136]** repose sur le principe de la décomposition successive des tâches. Contrairement à d'autres approches de planification, le concept central des HTNs ne sont pas des états, mais plutôt des

tâches. Un système de planification basée sur HTN décompose itérativement la tâche désirée en un ensemble de sous-tâches jusqu'à ce que l'ensemble résultant ne contienne que des tâches atomiques (primitives). Ces tâches peuvent être exécutées en appelant directement certaines opérations atomiques. A chaque itération de la décomposition des tâches, des testes sont effectués afin de vérifier si certaines conditions sont pas respectées (par exemple dépassement d'une certaine quantité de ressources disponibles) ou non. Le problème de planification est considéré comme résolu avec succès si la tâche complexe désirée est décomposée en un ensemble de sous-tâches primitives sans violer aucune des conditions données. SHOP2 **[137]** est un exemple de systèmes qui utilisent la planification HTN dans le domaine des services Web. SHOP2 se base sur la description des services en DAML-S pour effectuer une composition automatique de services. Le système SHOP2 appartient au projet SHOP (*Simple Hierarchical Ordered Planner*) dont l'objectif premier est de proposer une plate-forme de planification hiérarchique **[150]**. SHOP2 relie ce projet à la composition de services Web. D'autres travaux évoquent le fait d'utiliser la planification afin de composer les services Web, tels que **[151]**, **[152]**, ou encore **[153]**.

3.4.3.4. Réseaux de Petri

Les réseaux de Petri (*Petri Nets*) ont été introduits par C.A. Petri **[154]** pour la modélisation des systèmes concurrents. Un réseau de Petri est un graphe orienté, connecté, et bipartite dans lequel les nœuds représentent les places et les transitions, et les jetons occupent des places. Quand il ya au moins un jeton dans chaque place reliée à une transition, la transition est activée. Une transition activée peut tirer un jeton de toutes les places d'entrée, et le mettre dans chaque place de sortie. Plusieurs travaux se sont intéressés à la modélisation de la composition de services en utilisant les réseaux de Petri. Dans **[155]**, les auteurs réalisent une transformation des instructions BPEL en réseaux de Petri. Dans [**156**], les auteurs proposent un framework pour la conception et la vérification de la composition de services basée sur les réseaux de Petri. Dans **[138]**, les auteurs modélisent les services comme des réseaux de Petri en attribuant des transitions pour les méthodes et les places pour les états des services. A chaque service est associé un réseau de Pétri qui décrit son comportement. Dans ce réseau, un service dispose de deux ports: un port d'entrée et un port de sortie. À un instant donné, un service peut être dans l'un des états suivants : pas instancié, prêts, en cours d'exécution, suspendu ou achevé. Après la définition d'un réseau de Pétri pour chaque service, les opérateurs de composition (séquence, alternative, séquence non ordonnée, itération, parallèle, discriminateur, sélection et raffinement) réalisent la composition de services.

3.4.3.5. GraphPlan et SATPlan

Dans le modèle *GraphPlan*, le problème de composition de services est représenté par un quadruplet (P, W, r_{in}, r_{out}) tels que : P représente l'ensemble des paramètres d'entrées / sorties des services, W représente l'ensemble des services, r_{in} ($r_{in} \subset P$) représente l'état initial et r_{out} ($r_{out} \subset P$) représente l'état final (but). En se basant sur l'état initial, la planification *GraphPlan* construit un graphe de services en plusieurs étapes appelées « *time steps* » afin d'atteindre le but recherché. Une étape « *time step* » contient deux niveaux : le premier niveau exprime les services dont les préconditions sont vérifiées, et l'action de maintient de paramètres. Le deuxième niveau exprime les effets des services satisfaits et les paramètres maintenus. Une

fois le but recherché est atteint, *GraphPlan* procède par une recherche en arrière (*backward*) afin de trouver un plan valide de composition de services.

Le modèle SATPlan consiste à convertir le Graphplan en un ensemble d'expressions logiques comme proposé dans **[139]**. D'abord, l'état initial est exprimé par la conjonction de tous les paramètres satisfaits au niveau 0. Le but est exprimé par la conjonction de paramètres à satisfaire au niveau final. Ensuite, des expressions logiques expriment les relations entre les actions (services) et leurs préconditions ainsi que les relations d'inférence entre chaque fait et toutes les actions du niveau précédent. Enfin, les expressions logiques précédentes sont combinées dans une seule conjonction, comme une instance d'un problème de satisfiabilité (SAT), qui se résout par des méthodes appropriées telles que : la méthode complète qui se base sur des tables de vérité.

3.4.3.6. Processus de décision Markoviens

Dans l'article **[140]**, les auteurs proposent une nouvelle approche qui s'appuie sur les processus de décision Markovien (*Markovien Decision Process* : MDP) pour modéliser la nature dynamique et incertaine de l'environnement ainsi que le comportement stochastique des services Web. Le formalisme MDP utilisé dans cet article modélise la prise de décision comme un problème stochastique, de nature séquentiel et entièrement observable (avoir une perception complète de l'environnement qui est supposé accessible). Dans ce formalisme, le monde réel est modélisé par un ensemble d'états et d'actions. Une action (service) n'aboutit pas toujours au même état, puisque les actions ne sont pas déterministes (incertaines). L'introduction de la fonction de transition permet de raisonner sur ce type d'actions. Une fonction du coût (ou inversement, de récompense) est également introduite afin de spécifier le coût de l'exécution de chaque action à partir de chaque états. La solution du MDP consiste donc à trouver la meilleure politique selon le critère de performance considéré. La politique est une application qui, à un état du système, associe une action à effectuer. C'est simplement un passage (*mapping*) des états aux actions. La solution ainsi trouvée est une séquence de politiques indicées par le temps qui spécifie l'action optimale à exécuter pour chaque état. La solution est obtenue après une certaine période de temps appelée horizon. Si l'horizon est infini la politique obtenue est dite stationnaire, sinon la politique n'est pas stationnaire car la bonne action à exécuter peut dépendre du temps restant.

3.4.3.7. Systèmes multi-agents

Les agents sont des entités autonomes qui tentent d'atteindre un but commun, soit par collaboration ou à l'aide d'une entité centrale. Parmi leurs avantages c'est qu'ils peuvent utiliser un langage sémantique commun pour supporter des interactions complexes. Comme décrit dans **[141]**, les applications d'agents mobiles ont été initialement basées sur des composants, mais l'abstraction offerte par les langages sémantiques permet l'interaction entre les agents mobiles et les services. Les agents peuvent découvrir les services dont ils ont besoin pour l'invocation soit en recherchant dans un registre ou en demandant à d'autres agents. Certaines règles peuvent être également définies par l'utilisateur afin de guider le comportement des agents. Le principal avantage de cette approche est que les agents ne sont pas limités uniquement à l'exécution d'un workflow, mais ils peuvent suivre les actions de l'utilisateur et réagir aux différents changements. Plusieurs travaux ont exploité les

méthodologies agent et systèmes multi-agents (SMA) pour la composition de services. L'approche de **[157]** propose par exemple de composer des services à travers la coalition d'agents médiateurs qui ont pour rôle de réaliser les tâches requises par les services. Aussi, les travaux de **[107]** emploient un agent utilisateur qui publie dans un *blackboard* le but de composition à satisfaire. Les agents services existants récupèrent alors du *blackboard* les sous-buts à satisfaire, et vérifient s'ils peuvent ou non contribuer à une composition. L'ensemble des agents qui contribuent à la réalisation d'une même composition forment alors une coalition d'agents. Pour qu'un nouvel agent service rejoigne une coalition, il doit être sujet au vote de tous les agents services la formant.

3.4.4. Composition par approches sémantiques

Cette catégorie d'approches utilise les principes du Web sémantique pour fournir des compositions de services qui tiennent compte des propriétés sémantiques des services. Alors que d'autres approches identifient uniquement la structure des messages échangés, les approches sémantiques interprètent aussi le contenu de ces messages. Le Web sémantique propose d'ajouter plus d'information à la description des services Web afin d'automatiser plusieurs tâches comme la découverte de services, la négociation, la composition et l'invocation de services **[158]**. Dans ce qui suit, nous donnons un aperçu des approches pour la composition sémantique des services.

3.4.4.1. Annotation sémantique

L'annotation sémantique (*Semantic annotation)* marque la description des services Web avec des métadonnées qui correspondent aux ontologies. Les langages d'annotation sémantique tels que DAML+ OIL et OWL, sont utilisés pour la publication et le partage des données en utilisant des ontologies, tandis que les langages des services web sémantique tel que OWL-S (anciennement DAML-S) fournis une description sémantiquement enrichies pour les services Web. OWL-S utilise le profil de service (*Service Profile*), le modèle de processus (*Process Model*) et le modèle d'accès (*Grounding*) pour décrire un service Web. Le profil de service fournit les informations nécessaires pour un agent logiciel afin de découvrir un service, tandis que les modèles de processus et d'accès fournissent suffisamment d'informations pour un agent logiciel afin d'utiliser un service, une fois trouvé. La solution à base d'IA **[135]** introduit une annotation sémantique en utilisant DAML-S. L'initiative METEOR-S décrit un Framework pour l'annotation semi-automatique des services Web (MWSAF) **[142]**. Ce Framework propose plusieurs algorithmes utilisés dans la correspondance (*matching*) des schémas XML à des ontologies. Ces algorithmes examinent la similitude linguistique et structurée des concepts, puis ils sélectionnent celle qui offre la meilleure correspondance. L'architecture METEOR-S est composée de trois éléments: les bases d'ontologie (*Ontology-Store*) qui sauvegarde les ontologies existantes, la librairie de translation (*Translator Library*) qui effectue la transformation de *SchemaGraph* (représentation commune des schémas XML et des descriptions des ontologies) et la librairie de *matching* (*Matcher Library*) qui fourni les différents algorithmes de *matching*. WSDL-S, issus de METEOR-S, est un mécanisme permettant d'annoter sémantiquement des services Web. Il est maintenant la base pour les annotations sémantiques pour WSDL (SAWSDL)

[143]. SAWSDL décrit comment ajouter des annotations sémantiques pour les différentes parties d'un document WSDL.

3.4.4.2. Approches à base de règles

Les approches à base de règles (*Rule based approaches)* pour la composition sémantique définissent des règles spécifiques qui guident le processus de composition de services. Les auteurs dans **[144]** décrivent un modèle de composabilité qui vérifie quels sont les services Web qui peuvent interagir les uns avec les autres selon des règles de composabilité qui vérifient les propriétés syntaxiques et sémantiques des services. En se basant sur ce modèle de composabilité, une approche en quatre phases pour la composition automatique de services a été proposée: phase de spécification, phase de *matching,* phase de sélection et phase de génération. La phase de spécification permet une description de haut niveau des services composites à travers le langage de spécification de services composites (*Composite Service Specification Language: CSSL*), la phase de *matching* (*Matchmaking)* utilise les règles de composabilité pour générer des plans de composition en fonction des spécifications, la phase de sélection choisit le meilleur plan de composition et enfin la phase de génération fournit une description du service composite dans le langage de composition choisi.

3.4.4.3. Approches à base de connaissances

La composition basée sur la connaissance (*Knowledge based composition)* introduit des connaissances spécifiques à un domaine dans le processus de composition de services. Cette approche s'inspire du modèle de connaissances des systèmes d'aide à la décision qui se basent sur une utilisation intensive de la connaissance du domaine. Dans **[146]**, des connaissances spécifiques à un domaine sont utilisées pour guider la composition de services. Ces connaissances sont décrites dans des ontologies avec un vocabulaire communément admis et compréhensible. L'idée consiste à réutiliser des systèmes d'aides à la décision basés sur les connaissances, déjà décrites, dans le cadre d'un Framework orienté services pour l'informatique en grille (*Grid computing*). Ce Framework exploite des connaissances spécifiques à un domaine et fournit des recommandations pour spécifier des compositions de services sous forme de workflows. Pour ce faire, l'utilisateur donne d'abord une description initiale du problème et le système offre des suggestions compte aux services à composer.

3.4.4.4. Approches à base de politiques

Une politique est un moyen pour définir et modifier l'organisation et le comportement d'un système. Elle se compose généralement de règles spécifiant au système les actions à entreprendre pour répondre à une situation donnée. Plus formellement, une politique peut être définie selon deux perspectives **[159]** : des objectifs permettant de guider et déterminer les actions présentes et futures à exécuter au sein du système, et un ensemble de règles permettant d'administrer, superviser et commander l'accès aux ressources du système. Les approches de composition à base de politique (*Policy Based Approaches)* visent la composition de services tout en respectant les politiques définies pour différents contextes. Dans **[147]**, la couche de composition de services proposée prend en entrée les politiques et le domaine de connaissances de l'application. Les interfaces de services sont décrites soit dans un langage tel que WSDL ou OWL-S, ou bien écrites par un utilisateur. Un langage avec une syntaxe Java est utilisé à cet effet. L'élément principal de ce langage est les variables de liaison (*binding*

variables), qui fournissent des liens vers des services à distance en fonction de leur type et de leurs attributs de description. Les nouveaux objets ou les instances ne sont pas créés directement par le biais de ce langage, mais par des méthodes importées ou par le contexte d'exécution. L'utilisation de la syntaxe stricte de ce langage garantit que le service final composé obéit à la politique du contexte actuel. Dans **[145]**, en plus de la compatibilité syntaxique et sémantique, la compatibilité des politiques est également introduite dans la composition de services.

3.4.4.5. Dépendance d'entrées/sorties

Une approche qui exploite les dépendances d'entrée/sortie *(Input/Output Dependency)* est décrite dans **[148]**. Dans cette approche, chaque service est décrit par certains mots-clés, la liste de ses entrées et la liste de ses sorties. Chaque sortie peut être reliée soit à zéro, à une ou à une liste d'entrées, de telle sorte que les exigences d'entrée pour produire une sortie spécifique soient décrites. Le processus de composition de service commence par la création des listes de services qui ont des entrées similaires et produisent des résultats similaires. Si un service simple n'est pas trouvé, alors la recherche continue pour créer une liste de services qui prennent en entrée une partie des entrées initiales et éventuellement les sorties des services dans la liste précédente. De cette manière, un graphe d'invocation de services est créé. De même, dans les approches *FCoSC* (*Feasibility and Construction of Services Composition)* **[165, 166]**, et *CDSC (Context-aware Dynamic Service Composition)* **[167, 168]**, des graphes de composition de services sont construits automatiquement en exploitant les relations entre les entrées/sorties des services disponibles. Les graphes résultants sont considérés comme des plans de composition dans lesquels chaque tâche atomique correspond à un appel de service.

3.4.5. Composition par approches intergicielles

Cette catégorie d'approches se base sur un middleware (intergiciel) qui fourni des mécanismes de composition de services qui permettent aux applications de devenir adaptables, reconfigurables et tolérantes aux pannes. Les approches intergicielles permettent de résoudre un certain nombre de problèmes en relation avec la composition de services notamment la découverte et l'invocation des services. Bien qu'ils existent de nombreux types de middleware pour la composition de services, ils partagent tous deux parties communes. La première est un registre contenant des informations sur les services existants, tels que l'emplacement, les entrées et sorties. La deuxième partie est le moteur qui compose réellement les différents services disponibles et qui est responsable sur l'exécution de l'application à base de services. Dans cette section, nous nous limitons uniquement à certain middlewares qui supportent la composition de services en informatique ubiquitaire et ambiante. En outre, nous soulignons juste la vision globale de ces middlewares avant de détailler leur fonctionnement dans le **chapitre 4**.

Dans le middleware *PEIS-Ecologie* (*Physically Embedded Intelligent System)* **[160, 161]**, la composition de services est considérée comme un problème d'auto-configuration où des composants sont liés dynamiquement et composés dans des configurations significatives. Par la suite, une recherche est effectuée dans l'espace de toutes les configurations possibles afin de trouver une configuration admissible. Dans le middleware *WComp* (*Middleware for Ubiquitous Computing)* **[162]**, deux types de composition de services sont proposés: la

composition de haut niveau à l'aide des services SLCA (*Service Lightweight Component Architecture*), et la composition pour l'adaptation utilisant les aspects d'assemblage AA (*Aspects of Assembly*). Dans la première approche, SLCA (voir **section 3.4.2.4**) définit une architecture de services Web composites basés sur des événements. Ces services sont construits par assemblage de composants dans un conteneur. Dans la seconde approche, les aspects d'assemblage sont constitués par un tisseur (*weaver*) selon des règles logiques de fusion dans une spécification de haut niveau. Dans le middleware *SURF* (*Service-oriented Ubiquitous Robotic Framework)* **[163]**, la composition de services est basée sur le système SHOP2 qui utilise l'approche HTN **[164]**. Dans le middleware *MySIM* (*Spontaneous Service Integration for Pervasive Environment)* **[169]**, la composition de services est basée sur un appariement (*matching*) syntaxique ou sémantique entre les parties fonctionnelles des services. À partir de toutes les combinaisons obtenues, seuls les services qui sont équivalents en termes d'interfaces avec les services déjà existant dans l'environnement sont conservés. Dans le middleware *PERSE (Pervasive Semantic-aware Middleware)* **[93]**, les services et les tâches sont modélisés comme des conversations à l'aide d'automates à états finis (FSA). La composition de services est vue comme une intégration des conversations des services découverts afin de réaliser la conversation de la tâche cible de l'utilisateur. Dans le middleware *SeSCo* (*Seamless Service Composition)* **[92]**, la composition de services est basée sur la résolution de tâches. Tout d'abord, tous les services enregistrés sont analysés et stockés dans un graphe agrégé en fonction de la sémantique et la syntaxe de leurs paramètres. Par la suite, la résolution d'une tâche est effectuée en deux étapes. Dans la première étape, une ou plusieurs compositions possibles sont dérivées au niveau sémantique. Dans la seconde étape, les services qui peuvent prendre part à la composition sont identifiés en utilisant la syntaxe de leurs paramètres. Dans le middleware *URS* (*Ubiquitous Robotic Space)* **[170]**, le modèle de composition de services est basé sur un réseau d'interprétation du contexte CIN (*Context Interpretation Network*). Un CIN est une chaîne construite comme un service composite pour l'interprétation du contexte à partir des capteurs, des robots et des nœuds d'interprétation du contexte (ci). Dans le middleware *MUSIC* (*Middleware Support for Self-Adaptation in Ubiquitous and Service-Oriented Environments)* **[171]**, les plans de composition de services sont prédéfinis et décrits dans un modèle sensible à la QoS (*QoS-aware model*) d'une manière abstraite. La planification consiste alors à sélectionner le meilleur plan de composition de services en termes d'utilité pour l'utilisateur final. Dans le middleware *VRESCo* (*Vienna Runtime Environment for Service-oriented Computing*) **[172]**, un service est défini par ses caractéristiques fonctionnelles et non fonctionnelles. Initialement, la structure d'un service composite est spécifiée manuellement par l'utilisateur. Par la suite, le problème de sélection de services est formulé comme un problème de satisfaction de contraintes (CSP), qui prend comme entrées tous les services candidats regroupés par fonction. Dans le middleware *SeGSeC* (*Semantic Graph-Based Service Composition*) **[173]**, les composants sont représentés à l'aide des graphes sémantiques qui comportent des nœuds et des liens marqués. L'approche de composition de services proposée crée un *workflow* en examinant sa sémantique avec la requête de l'utilisateur. Dans le middleware *MEDUSA* **[174],** les auteurs supposent que chaque environnement ubiquitaire fournit un ensemble de cartes associées aux instances des services disponibles dans l'environnement. En outre, le plan de composition de service (application) est conçu manuellement de manière abstraite en utilisant un outil dédié à la composition. Les

services constituant l'application conçue sont connectés au moment de l'exécution aux instances des services disponibles dans l'environnement par le compositeur d'application et contrôlés par les utilisateurs grâce à un ensemble d'interfaces utilisateur.

3.5. Adaptation des services au contexte

Nous avons étudié dans la partie précédente les différentes approches de composition de services. Cependant, le manque de dynamisme reste une des limites majeures de ces approches. Afin de pallier à cette limite, certaines approches se dotent d'un ensemble de mécanismes qui leur permettent d'adapter leur comportement en fonction des changements du contexte. Dans cette section, nous étudions l'adaptation des services au contexte qui est un moyen efficace pour assurer le dynamisme des approches de composition de services. Plusieurs auteurs dans la littérature se sont penchés sur la technologie des services Web et l'adaptation au contexte. Certains travaux comme **[175], [176], [177]** considèrent qu'une nouvelle préoccupation qui met la notion de contexte au centre des travaux sur les services Web émerge. Ceci est dû en fait, à la nature dynamique d'Internet en général et des services Web en particulier. D'une part, les services Web ont besoin d'adapter leurs opérations afin de répondre au mieux à la situation dans laquelle ils sont appelés. D'autre part, les services doivent pouvoir participer ou non à une composition en fonction du contexte **[178]**. Cependant, les langages et les spécifications relatives aux services Web, notamment le langage WSDL ou encore le langage OWL-S, ne prennent pas en compte la notion de contexte **[179]**.

3.5.1. Notion de contexte

3.5.1.1. Définitions du contexte

Au regard de la littérature informatique, il existe plusieurs définitions à la notion de « contexte » **[180, 181].** Cette notion a suscité beaucoup d'intérêt de plusieurs chercheurs dans le domaine de l'informatique pervasive ou ambiante. De nombreux travaux issus de cette communauté de recherche tentent de donner une définition du contexte. Selon **[182]**, quatre familles de contexte peuvent être identifiées. Chacune de ces familles apporte sa propre définition du contexte. Ces familles de contexte sont les suivantes : *contexte géolocalisé*, *contexte unifié*, *contexte généralisé* et *contexte environnemental*. (1) La notion de *contexte géolocalisé* défini le contexte en se référant principalement à la géolocalisation de l'utilisateur. Les auteurs dans **[183]** définissent le contexte comme étant « *la localisation de l'utilisateur, les identités et les états des personnes et des objets qui l'entourent* ». Pour ces auteurs, étudier le contexte c'est répondre aux questions : « Où es-tu ? », « Avec qui es-tu ? », « De quelles ressources disposes-tu à proximité ? ». (2) La notion de *contexte unifié* introduite dans **[184]** spécifie le contexte comme étant une situation donnée par les réponses à six questions : Qui (*Who*) ? Quoi (*What*) ? Où (*Where*) ? Quand (*When*) ? Comment (*How*) ? Pourquoi (*Why*) ? (3) La notion de *contexte généralisé* propose une définition générale du contexte. Par exemple, dans **[185]**, le contexte est définit comme «*l'ensemble d'informations structuré et partagé qui évolue et sert l'interprétation* ». De même, dans **[186],** le contexte est défini comme étant ce qui entoure et donne du sens à quelque chose d'autre. (4) La notion de *contexte environnemental* relit le contexte aux éléments de l'environnement de l'utilisateur

tout en, introduisant de nouveau concepts comme l'heure, la saison, la température, l'identité et la localisation de l'utilisateur. Dans **[187]**, le contexte est défini comme «*les éléments de l'environnement dont l'ordinateur à connaissance*». Dans le même cadre, les auteurs dans **[188]** présentent une définition du contexte qui introduit explicitement la notion du temps : «*C'est la localisation, l'environnement, l'identité et le temps relatif à un utilisateur* ». La définition la plus largement adoptée dans le domaine de l'informatique sensible au contexte est celle de Dey **[189]** : « *le contexte couvre toutes les informations pouvant être utilisées pour caractériser la situation d'une entité. Une entité est une personne, un lieu, ou un objet qui peut être pertinent pour l'interaction entre l'utilisateur et l'application, y compris l'utilisateur et l'application eux-mêmes* ».

La plupart des chercheurs **[190, 191, 185]** ont fait le consensus sur les constats suivants : (i) il n'y a pas de contexte sans contexte : le contexte n'existe pas en tant que tel, il est défini en fonction des objectifs à atteindre qui représentent sa finalité (utilité). (ii) Le contexte est un espace d'information qui sert l'interprétation : la capture du contexte n'est pas une fin en soi mais elle doit servir certains objectifs. L'interprétation du contexte dépend alors de sa finalité. (iii) Le contexte est un espace d'informations partagé par plusieurs acteurs : ces acteurs peuvent concerner l'environnement, l'utilisateur, le service,…etc. (iv) Le contexte est un espace d'information évolutif : le contexte n'est pas figé, mais il se construit au cours du temps. D'après ces quatre axes de définition du contexte, nous déduisons qu'il est important de bien cerner et spécifier la finalité du contexte et définir les informations nécessaires et suffisantes pour servir cette finalité.

En informatique ambiante, le contexte physique est d'abord acquis à partir de données de capteurs disséminés dans l'environnement. Ces données sont, par la suite, élaborées en un format plus adapté à leur utilisation. Par rapport à la donnée brute produite par le capteur, la donnée traitée est moins précise mais plus adaptée **[192]**. Le contexte en informatique ambiante est très souvent formé autour des besoins d'une application particulière et d'un domaine spécifique.

3.5.1.2. Représentation du contexte

Afin de mieux gérer et manipuler le contexte, une application doit pouvoir organiser les données contextuelles capturées et les stocker pour une utilisation future. Ceci passe inévitablement par la modélisation et la représentation du contexte. Au regard des caractéristiques variées des données contextuelles telles que l'hétérogénéité, la dynamicité et l'imperfection, il est indispensable de définir un modèle pour décrire ces données. Dans la littérature, nous distinguons trois types d'approches principales pour la modélisation du contexte : approche paires/triplet, approche orientée modèle, et approche orientée ontologie. Cette classification se base sur la structure de données utilisée pour modéliser le contexte, les outils utilisés à cet effet, l'expressivité du modèle et la possibilité de le réutiliser.

- **Approche paires/triplet:** c'est l'une des premières approches sur la modélisation du contexte qui est introduite par Schilit et al. **[193]**. L'information de contexte est modélisée sous formes de paires (attribut, valeur). L'attribut représente l'identifiant du contexte, et la valeur représente les informations associées à ce contexte. Par exemple la liste des noms des occupants d'une salle est la suivante : *OCCUPANTS = adams :schilit :theimer :weiser*

:welch. Le contexte est également modélisé par **[194]** comme un triplet constitué de l'attribut, de la valeur observée du contexte, et un degré de certitude sur la cohérence des données observées. Par exemple, (*Location*, *ConferenceRoom*, 95), définit la localisation comme le lieu d'une conférence avec un degré de certitude égal à 95.

- **Approche orientée objet :** cette approche proposée par Henricksen et al. **[195]** se base sur un ensemble de concepts du paradigme orienté objet. Les informations de contexte sont regroupées en un ensemble d'entités. Chaque entité représente un objet conceptuel ou physique tel qu'une personne, un dispositif ou un réseau. Les propriétés des entités telles que le nom d'une personne sont représentées par des attributs. Les entités sont liées à leurs attributs à travers des associations. Dans le même cadre, les auteurs dans [**196]** proposent une extension du langage UML appelé *ContextUML* afin de modéliser le contexte.
- **Modélisation par une ontologie :** les techniques de représentation de données proposées dans les langages à objets comme Java et C++ ne garantissent pas un niveau d'expressivité suffisant pour décrire les différentes classes du contexte. Pour ce faire, il est nécessaire de s'appuyer sur des techniques de représentation des connaissances avancées telles que les ontologies. Ces dernières offrent en effet un haut niveau d'expressivité pour construire un modèle du contexte disposant d'un vocabulaire riche et extensible. Dans une ontologie, les connaissances d'un certain domaine sont représentées formellement comme un ensemble d'objets ayant des relations entre eux. La modélisation du contexte par une ontologie a un réel intérêt ; elle permet le partage des informations de contexte dans un système distribué, et avec une sémantique bien définie, elle permet l'utilisation d'agents intelligents pour faire un raisonnement sur ce contexte. Plusieurs modèles d'ontologies ont été proposés afin de modéliser le contexte. *CoBrA-ONT* (*Context Broker Architecture ONTology*) est une ontologie écrite en utilisant le langage OWL-DL **[179]**. Cette ontologie a été créée pour décrire le contexte que les applications multi-agents de maison intelligente utilisent **[197]**. Les langages d'ontologies OWL et OWL-S sont également utilisés pour la modélisation du contexte **[198]**. *CONON* (*CONtext ONtology*) est une ontologie de contexte extensible écrite en utilisant le langage OWL-DL. Elle permet de décrire le contexte auquel des services peuvent être sensibles dans un environnement ubiquitaire **[199]**. Actuellement, l'approche orientée ontologie est l'approche la plus expressive et la plus adopté pour la description du contexte dans un environnement ambiant.

3.5.1.3. Acquisition des informations du contexte

L'acquisition des informations de contexte constitue la première étape dans le processus de traitement du contexte. Concrètement, le contexte fourni à une application peut être acquis à partir des sources diverses. Prenons le cas, par exemple, d'un utilisateur mobile qui cherche le plus proche restaurant qui répond à ses préférences. L'utilisateur doit spécifier ses menus préférés et éventuellement d'autres préférences telles que le coût. Le système doit pouvoir répondre aux exigences de l'utilisateur en se basant sur les informations clairement spécifiées par l'utilisateur et sa position actuelle fournie par le système GPS, qui sera transformée en une adresse contenant le pays, la ville,…etc. Cet exemple nous illustre les différentes sources du contexte. D'abord, les préférences de l'utilisateur sont fournies explicitement par l'utilisateur au système. Ensuite, les coordonnés de l'utilisateur sont capturées par le dispositif de

positionnement GPS. Enfin, ces coordonnés sont transformé en une adresse. Ainsi, Mostéfaoui et al. **[200]** distinguent trois types de contexte selon les méthodes utilisées pour collecter l'ensemble des informations de ce contexte :

- **Contexte capturé** : il s'agit du contexte qui est acquis directement par les capteurs physiques disséminés dans l'environnement telles que les capteurs de température, de bruit, GPS, les caméras, …etc. Ce type de contexte peut être également acquis par des capteurs logiciels, par exemple, les mesures des pertes ou de la gigue lors de la transmission de la vidéo sur des liaisons sans fil peuvent constituer le contexte d'un algorithme de régulation du débit. Toutefois, le contexte collecté à partir des capteurs est sujet à de fréquents changements. Sa collecte nécessite une interaction avec des capteurs logiciels ou matériels distribués et hétérogènes.
- **Contexte explicitement fourni** : par exemple, les informations du profil d'un client sont des informations explicitement fournies par ce dernier, elles sont caractérisées par un changement peu fréquent. Comme nous l'avons cité dans l'exemple précédent, l'utilisateur communique explicitement ses préférences au système (application).
- **Contexte dérivé ou interprété** : il s'agit d'un contexte de haut niveau qui est déduit à partir des informations de contexte de bas niveau (contexte capturé). Dans l'exemple précédent, le pays et la ville où se trouve l'utilisateur représentent un contexte de haut niveau déduit à partir du contexte de bas niveau qui représente ses coordonnées collectées à partir d'un GPS.

3.5.2. Notion de sensibilité au contexte

3.5.2.1. Définition de la sensibilité au contexte

Dans la littérature, il existe beaucoup de synonymes au terme « sensible au contexte ». On parle de systèmes « adaptable au contexte », « réactifs », « dirigés par l'environnement » ou « qui prend en compte le contexte ». Ces appellations sont toutes proches de la notion de sensibilité au contexte et proviennent du terme anglais « *context-awareness* » qui a été introduit pour la première fois par Schilit et Theimer en 1994 **[183]**. Ces auteurs ont défini ce terme comme étant un logiciel qui « *s'adapte en fonction de sa localisation d'utilisation, de l'ensemble des personnes et des objets à proximité, ainsi que de la variation de ces objets à travers le temps* ». Nous constatons que cette définition porte uniquement sur quelques variables tels que la localisation, l'identité des personnes et des objets voisins ainsi que les changements sur ces objets. Un peu plus tard, d'autres définitions ont vu le jour. Brown **[201]** souligne qu'une application est sensible au contexte si elle peut effectuer des actions en fonctions du contexte utilisateur détecté par les capteurs. Dey et Abowd **[189]** proposent qu' « *un système est sensible au contexte s'il utilise le contexte pour fournir des informations pertinentes ou des services utiles à l'utilisateur, l'utilité dépend de la tâche de l'utilisateur* ». Dans **[202]**, la sensibilité au contexte est définie comme étant la capacité du système à exploiter le contexte pour fournir à l'utilisateur des services pertinents. Cette notion de sensibilité au contexte a fait émerger une nouvelle tendance de l'informatique appelée informatique sensible au contexte (*context-aware computing*). Cette dernière caractérise un paradigme de l'informatique dans lequel les applications perçoivent la situation de l'utilisateur dans son environnement et adaptent ainsi leurs comportements **[203]**.

L'adaptation de l'application aux informations du contexte est illustrée par la **figure 3.13**. Cette figure montre clairement que les éléments de contexte agissent sur le comportement de l'application. De même, il est possible pour l'application d'agir et de modifier le contexte. C'est ce qui est appelé boucle de contexte où la sortie contextuelle de l'application va modifier les valeurs des variables contextuelles.

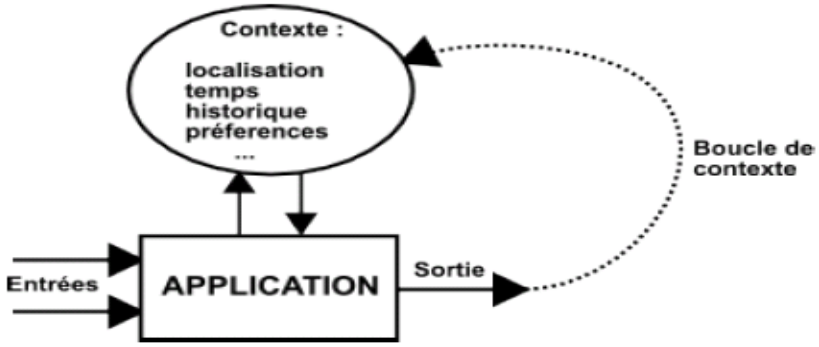


Figure 3.13 : Application tenant compte du contexte

Lorsque le contexte est modifié, l'application doit être ajustée pour qu'elle soit à même de réaliser son objectif au mieux dans ces nouvelles conditions. Une adaptation est ainsi une *modification* d'un système, suite à la *décision* d'un ajustement en réponse à *un changement dans son contexte* **[204]**. L'adaptation constitue le dernier maillon de la chaîne de sensibilité des systèmes au contexte. D'autres auteurs comme **[205]** affirment que d'une manière générale l'adaptation d'une application suit un processus à trois étapes successives : une étape de déclenchement, une étape de décision et une dernière étape de réalisation.

3.5.2.2. Mécanismes d'adaptation au contexte

La modélisation des informations de contexte est primordiale pour leur gestion à savoir leur transport, leur échange et leur stockage. Mais vu la diversité des informations de contexte, cette modélisation est étroitement liée au domaine de leur utilisation. Il existe plusieurs mécanismes qui permettent aux applications d'être adaptables et réagir ainsi à des changements de contexte. Les mécanismes les plus connus **[206] [207]** sont les suivants:

- **Réflexivité :** Le concept de réflexivité a été introduit par Smith en 1982 **[208]**, et depuis il a été largement utilisé pour l'adaptation des applications à base de composants. Selon **[208]**, la réflexivité représente la capacité d'un système à s'observer et à agir sur lui-même durant son exécution. Un système réflexif doit présenter deux niveaux : un niveau de base et un niveau « méta ». Le niveau de base comporte les données et le code fonctionnel du système (le « quoi » du système) et le niveau « méta » comporte les données et le code non fonctionnel du système (le « comment » du système). Le niveau méta permet au système de s'observer et d'agir sur lui-même pour s'adapter. Ces deux niveaux sont fortement connectés, c'est-à-dire que tout changement dans l'un d'eux se répercute sur l'autre. Les interactions entre les deux niveaux se font dans les deux sens. Ainsi, **[208]** identifie deux aspects : l'introspection et l'intercession. L'introspection est la capacité d'un système à examiner son propre état et l'intercession, est sa capacité à modifier son propre état d'exécution ou d'altérer sa propre interprétation [**178**].

- **Programmation par aspects :** Le paradigme de programmation par aspect a été mis en place afin d'élargir la possibilité de réutilisation du code, puisqu'il permet de développer des applications en séparant leur code métier de leur code technique (non fonctionnel) selon le principe de séparation des préoccupations (*Separation of Concerns*) **[209]** dont l'objectif est de permettre la maîtrise de systèmes logiciels en adressant, de façon isolée, les diverses préoccupations qui les composent. Ceci rend le code indépendant de l'infrastructure utilisée. Cette séparation permet de structurer l'application en modules indépendants représentant différents aspects : un noyau qui représente le cœur fonctionnel de l'application (le « quoi » de l'application); plusieurs aspects qui représentent des propriétés transversales au noyau (le « comment » de l'application). La construction d'une application en utilisant ces différents modules nécessite une étape d'intégration du noyau avec les différents aspects. Cette intégration se traduit par l'établissement d'une jonction qui se situe au niveau d'un ensemble de points du flot d'exécution du noyau, appelés points de jonction. Ce type de programmation peut aussi être utilisé comme technique d'adaptation. L'intégration d'un procédé d'adaptation dans une application est considérée comme du code technique. De nouveaux concepts ont été développés et définis pour la programmation par aspect **[210]** : aspect, greffon (*advice*), points de jonction (*join points*), points de coupure ou d'action (*pointcuts*) et tisseur (*weaver*). Un *aspect* regroupe un code (*advice*) que l'on souhaite greffer (ou tisser) au sein d'un code source grâce à un programme appelé tisseur (*weaver*) ainsi que des points de coupure ou d'action (*pointcuts*). Un *greffon* (*advice*) correspond au code qui sera activé à un certain point de jonction. *Les points de jonction* (*join points*) correspondent à des points précis dans l'exécution d'un programme tels que l'appel d'une méthode et l'affectation d'une variable, où on peut insérer un *advice*. Un *point de coupure ou d'action* (*pointcut*) définit un ensemble de points de jonction (*join points*) satisfaisant aux conditions d'activation de l'aspect. En d'autres termes, un *pointcut* définit les différents emplacements où un *advice* particulier sera inséré. Un *tisseur* (*weaver*) correspond à un programme qui permet de tisser le code des aspects dans le code des méthodes des classes. Ce tissage peut avoir lieu à la compilation ou à l'exécution (appelé aussi tissage dynamique).
- **Moteurs d'inférence :** Lorsque les informations de contexte sont représentées à l'aide d'une ontologie, l'intelligence d'un système sensible au contexte peut être implémentée à l'aide d'un moteur d'inférence qui raisonnera sur les informations de contexte. Un moteur d'inférences est implémenté par un ensemble de faits et de règles appliquées sur les informations de contexte collectées à partir des capteurs. Parmi les implémentations existantes de moteurs d'inférences, nous pouvons citer Flora2 **[179]** et Jena 2 **[211]**.

3.5.3. Approches d'adaptation de la composition de services

Dans cette section, nous allons présenter les approches d'adaptation de la composition de services. De nombreux travaux intègrent, soit lors de la définition de la composition, soit lors de l'exécution de la composition de services, le concept d'adaptation au contexte. Nous étudions ici trois techniques d'adaptation de la composition de services à savoir: adaptation par sélection de services, adaptation par tissage d'aspects (ajout de nouvelles préoccupations) et adaptation par apprentissage.

3.5.3.1. Adaptation par sélection de services

Le processus de composition de services peut générer et fournir plusieurs plans ou workflows pour un même service composite. Cela dépend du nombre de services disponibles et leurs capacités offertes. Par exemple, plusieurs imprimantes peuvent être candidates pour la réalisation de la tâche d'impression. Ici, les imprimantes candidates sont considérées comme des services fonctionnellement équivalents. L'introduction d'un mécanisme de sélection de services permet de choisir les services les plus appropriés et les plus adaptés à un contexte donné. Ainsi, fournir aux utilisateurs de meilleurs services devient une caractéristique intéressante pour les mécanismes de composition de services. La sélection de services est basée sur l'évaluation de chaque service individuel et la combinaison globale des services. Dans les environnements ambiants, cette évaluation dépend fortement de nombreux critères tels que les informations non fonctionnelles d'un service fournies comme des propriétés de qualité de service (QoS) et les informations additionnelles comme le contexte de l'utilisateur et de l'application, la qualité du réseau, et ainsi de suite. Comme souligné dans les travaux **[149, 212, 213, 214],** les informations de contexte sont utiles pour le choix des services. Les requêtes des utilisateurs peuvent ainsi être dirigées vers les services qui présentent une meilleure qualité de service **[149, 215, 216]**. Saif et al. **[217]** intègrent les informations contextuelles dans un mécanisme de planification dans un environnement intelligent. Les auteurs indiquent que la manière pour satisfaire un objectif particulier de la planification peut dépendre du contexte. Mokhtar et al. **[218]** présentent une approche de composition de services sensible au contexte. Cette approche se base sur l'intégration des workflows dans un environnement ubiquitaire. Maamar et al. **[219]** proposent une approche de composition de services basée sur les systèmes multi-agents (SMA) et les systèmes sensibles au contexte pour fournir à l'utilisateur le meilleur service dans un environnement pervasif. Hassine et al. **[220]** proposent une formalisation générique du problème de composition de services basés sur le problème d'optimisation de contraintes (COP). Cette formalisation est basée sur l'intervention incrémentale de l'utilisateur afin de trouver la composition optimale de services selon des critères prédéfinis au moment de l'exécution. Qiu et al. **[221, 222]** se sont basés sur les diagrammes d'états (*statecharts*) et les graphes acycliques directs DAG (*Direct Acyclic Graph*) afin de représenter les plans de composition de services qui sont construits par l'approche de planification HTN. Dans leur travail, les auteurs proposent une extension d'OWL-S, nommée OWL-SC (*OWL-S Context*) permettant d'intégrer le contexte à la description des services Web. Les auteurs distinguent trois catégories de contexte : le contexte de l'utilisateur (*U-Context*), le contexte du service Web (*W-Context*) et le contexte de l'environnement (*E-Context*). Chaque catégorie de contexte est représentée par une ontologie et est intégrée aux ontologies existantes d'OWL-S. Nahrstedt et al. **[223]** utilisent l'idée des algorithmes de planification globale pour la composition dynamique de services. Les auteurs proposent un modèle qui calcule un plan initial. Par la suite, ce plan est révisé si nécessaire lors de l'exécution en fonction du contexte. Doshi et al. **[140]** proposent une approche de sélection de services basée sur le formalisme MDP afin de tenir compte de la dynamique et du caractère incertain de l'environnement. Dans cette approche, les services composites sont décrits sous forme d'un workflow. Alirifai et al. **[224]** explorent une technique de programmation linéaire pour la sélection de services. Ben Mabrouk et al. **[225]** proposent des heuristiques pour estimer la valeur globale d'un service composite représenté sous forme

d'une conversation. Michlmayr et al. **[226]** indiquent que les attributs non fonctionnels de qualité de service (QoS) tels que le temps de réponse sont nécessaires pour permettre la sélection de services. Les auteurs proposent un environnement d'exécution des services appelé VRESCo (V*ienna Runtime Environment for Service-Oriented Computing*). Cet environnement intègre la QoS et d'autres métadonnées dans la description des services. Dans la suite du même travail, Leitner et al. **[227]** intègrent un mécanisme de rétroaction basé sur la sélection de services dans VRESCo. Ce mécanisme capture la qualité d'expérience QoE (*Quality of Experience*) perçue par les utilisateurs finaux sur un service. La QoE est modélisée en utilisant les commentaires des différents utilisateurs antérieurs d'un service. Ainsi, la QoE est utilisée pour affiner et classer les services disponibles. Ferreira et al. **[228]** soulignent que la consommation d'énergie est également un problème de qualité de service. Les auteurs présentent une nouvelle technique sensible à l'énergie et à la qualité de service afin de résoudre le problème de la sélection et la concrétisation des services.

3.5.3.2. Adaptation par tissage d'aspects

Comme nous l'avons souligné dans la **section 3.5.2.2**, la programmation par aspects est un paradigme qui permet de séparer entre le code métier (code de base) d'une application et celui des préoccupations transverses (sécurité, monitoring, etc.). L'idée de la programmation par aspects est donc de représenter de manière séparée, dans des aspects, ces préoccupations transverses. Par la suite, l'injection du code de ces préoccupations dans le code de base se fait par l'intermédiaire d'un tisseur d'aspects. Plusieurs travaux ont intégrés les aspects pour l'adaptation des applications. Par exemple, les aspects ont étés couplés aux paradigmes services **[229, 230, 231, 232]** pour l'adaptation dynamique des compositions de services. Leurs travaux s'intéressent aux deux principaux problèmes du langage BPEL à savoir la modularité et les adaptations dynamiques de l'orchestration définie. En BPEL les orchestrations sont prédéfinies et statiques, il n'est pas possible de modifier une orchestration dynamiquement à l'exécution. Les auteurs considèrent que la programmation par aspects peut résoudre ces problèmes grâce au tisseur qui peut apporter la dynamicité nécessaire par l'ajout de ce qui est défini comme de la modularité transverse, c'est à dire la séparation des services d'une application et de ses propriétés non fonctionnelles (sécurité, persistance ...). Dans ses travaux de thèse **[233, 234]** a traité les problématiques de l'adaptation dynamique du service ainsi que l'interfaçage correct entre le client et le service. Les auteurs ont proposé une approche basée sur la programmation orientée aspect qui permet de changer le comportement d'un service Web au moment de l'exécution **[233, 234, 235].** Dans d'autres travaux, les aspects ont étés couplés aux paradigmes composants. Par exemple FAC (*Fractal Aspect Component*) **[236]** intègre la notion d'aspects dans Fractal **[237]**. L'adaptation consiste en des modifications de la structure d'un assemblage de composants. Il propose alors la définition de la notion de composants encapsulant une préoccupation transverse appelés *Aspect Component* (AC). Le code du composant représente alors le greffon de l'aspect et les points de coupe sont définis au niveau du connecteur. Les points de jonction sont les composants, les interfaces et les méthodes sur lesquels un AC peut être appliqué. L'application d'un AC peut se faire à l'exécution. Le *Middleware* WComp **[162]** réalise une composition pour l'adaptation utilisant les aspects d'assemblage AA (*Aspects of Assembly*). Ces aspects d'assemblage sont constitués par un tisseur (*weaver*) selon des règles logiques de fusion dans une spécification de haut

niveau. Dans **[122]**, les auteurs proposent un modèle architectural ainsi qu'un mécanisme d'adaptation qui, à partir du principe de séparation des préoccupations permet d'exprimer la variabilité du système. Le mécanisme d'adaptation proposé, appelé cascade d'aspects, est expérimenté en se basant sur les Aspects d'Assemblages et la plateforme d'exécution WComp. Dans l'article **[238]**, les auteurs présentent un système sensible au contexte appelé ORCA (*a lOw Response time and dynamic Context Adaptation rule system*) pour répondre aux besoins des applications ubiquitaires qui cherchent à adapter leur comportement aux changements dynamiques et imprévisibles du contexte perçu dans l'environnement. La sensibilité au contexte est gérée par des règles comme une préoccupation non fonctionnelle définie séparément des adaptations du noyau fonctionnel des applications, mais elle est traitée dans le même processus. Les règles définies pour réaliser la sensibilité au contexte sont internalisés dans les applications en utilisant l'adaptation de la composition. Ces règles sont intériorisées, de manière réactive, en fonction du contexte disponible et découvert. A chaque observateur du contexte est associé un ensemble de règles pour la détection d'une situation. Ainsi, l'adaptation de la composition est déclenchée lorsque l'observateur défini par certaines règles relatives à la situation actuelle devient disponible.

3.5.3.3. Adaptation par apprentissage

Les adaptations par sélection de services ou encore par tissage d'aspects sont en général déclenchées lors du changement de contexte ou la découverte d'un nouveau contexte pertinent. Ces adaptations sont alors dites réactives, et ce, à l'inverse des adaptations proactives qui préparent de nouvelles actions pour les appels à venir lors de changements de contexte ou la détection d'un nouveau contexte pertinent. Ce type d'adaptation permet alors d'anticiper la détection d'un contexte pertinent grâce à un historique des contextes observés ou par un mécanisme d'apprentissage comme le fait Amigo **[239]**. L'utilisation des techniques d'apprentissage sont envisagée dans plusieurs travaux afin de réaliser des adaptations proactives. La phase d'apprentissage peut être couplée à une phase d'évaluation du contexte fin de réaliser une adaptation. La phase d'évaluation peut être vue comme la détection, la reconnaissance d'une situation déclenchant une adaptation de l'application [**240**]. Par exemple, dans AMIGO **[241],** la phase d'apprentissage est réalisée avant la phase d'évaluation tandis qu'Aura **[242, 243]** la place après. SOCAM **[244]** réalise également cette phase d'apprentissage mais l'adaptation proprement dite est laissée aux applications qui doivent la réaliser. Par exemple, Dehousse et al. **[245]** introduisent l'apprentissage par renforcement dans un mécanisme de sélection de services. Afin de mettre en place des systèmes de composition de services auto-adaptatifs, des mécanismes d'apprentissage par renforcement peuvent être utilisés **[246, 247].** Comme le souligne Gaber **[246]**, ces systèmes auto-organisant peuvent être inspirés, par exemple, à partir d'un système biologique, comme le système immunitaire naturel. Dans ce cadre, Itao et al. **[248]** proposent une plate-forme appelée JaNet pour le développement d'applications réseau à grande échelle, distribués, hétérogènes et dynamiques. L'adaptabilité de JaNet est inspirée d'un grand système biologique, en l'occurrence la colonie d'abeilles. Plus précisément, un service est mis en œuvre par un ensemble d'agents distribués, appelé cyber-entités. Ces services sont crées de manière adaptative en fonction des préférences de l'utilisateur **[249]**. Ceci est analogue à une colonie d'abeilles composée de plusieurs abeilles (cyber-entités). Chaque cyber-entité

implémente un composant fonctionnel lié à son service ou application. En outre, une cyber-entité a des comportements simples tels que la migration, la réplication, la reproduction, et la mise en relation, et met en œuvre un ensemble d'actions relatives à un service qu'elle fournit. Une autre approche à base d'agents, ayant des capacités d'auto-adaptation et d'auto-organisation, a été proposée dans **[247, 250, 251]**. Cette approche, appelée *ImmuneNet*, est inspirée par le système immunitaire humain. *ImmuneNet* fournit un système évolutif et adaptatif pour la découverte et la composition de services dans des environnements ubiquitaires. Dans cette approche, les serveurs sont organisés en communautés multi-agents décentralisés. Plus précisément, une communauté représente un service composite. Un service composant contient un ensemble de ressources matérielles ou logicielles que les utilisateurs doivent découvrir et sélectionner.

3.6. Conclusion

Nous avons présenté dans ce chapitre un aspect complexe des approches à services - la composition de services - auquel nous nous intéressons dans ces travaux. La composition de services permet l'intégration et la réutilisation des services dans diverses applications. Cette intégration peut se faire de plusieurs manières résultant en un ensemble de catégories de composition de services : manuelles, semi-automatiques ou automatiques tout en implantant des compositions statiques, dynamiques ou encore adaptatives avec un contrôle centralisé, hiérarchique ou distribué. Techniquement, ces différentes catégories peuvent être réalisées par plusieurs approches de composition de services. Nous avons identifié cinq approches principales: la composition par procédés, la composition structurelle, la composition par planification, la composition sémantique, et la composition par intergiciels. Pour chaque approche, nous avons illustré un ensemble de techniques qui sont adoptées afin de réaliser concrètement une composition de services. Cependant, afin d'améliorer les résultats de la composition de services et assurer son adaptation au contexte d'utilisation, ces approches doivent incorporer certains mécanismes d'adaptation au contexte. Nous avons souligné trois mécanismes principaux : la sélection de services, le tissage d'aspects et l'apprentissage pour l'adaptation. Ces mécanismes qui peuvent être intégrés d'une manière réactive ou proactive, permettent de rendre une application sensible et adaptable au contexte. Toutefois, il est important de noter que peu de travaux réalisent des adaptations proactives.

Nous ne considérons pas qu'une approche de composition de services identifiée soit meilleure qu'une autre. En effet, le choix de privilégier une approche par rapport à une autre dépend fortement du domaine et du contexte pour lesquels la composition de services est réalisée. Compte tenu des travaux de cette thèse qui s'inscrivent dans le domaine de l'intelligence ambiante, le choix d'une approche de composition de services est influencé par les réponses fournies à la réalisation des exigences des environnements ambiants dans un cadre orienté services (**voir section 2.6.2 du chapitre 2**). Ainsi, afin de distinguer entre les différentes approches de composition de services en termes de leur adéquation au domaine de l'intelligence ambiante, nous identifions trois discriminants importants à savoir : l'automatisation, l'adaptation et la complétude.

Automatisation : le degré d'automatisation du processus de composition de services se réfère au degré d'intervention humaine dans la spécification des services composites. Dans les environnements ambiants, dynamiques et à large échelle, il est pratiquement impossible pour un utilisateur de spécifier manuellement les services composites. En effet, face à un nombre important de services qui changent fréquemment, la tâche de l'utilisateur devient délicate et fastidieuse. Comme nous l'avons souligné dans la **section 2.6.2.5 du chapitre 2**, une automatisation élevée du processus de génération des services composites réduit considérablement l'intervention humaine et incrémente ainsi l'intelligence et la transparence du système. Les approches de composition par planification semblent êtres les mieux placées pour répondre à cette exigence. En effet, dans ce type d'approche, l'intervention de l'utilisateur est réduite à la spécification de l'état initial et du but à atteindre. Par la suite, le planificateur se charge de trouver un enchainement approprié de services qui réalise l'objectif spécifié. Toutefois, ces approches nécessitent une description standard et intelligible des services afin qu'ils soient interprétables par des machines. Les approches sémantiques offrent des réponses prometteuses quant aux formats de description des services. Ces formats, qui se basent essentiellement sur la notion d'ontologie, décrivent les services dans une manière compréhensible et les préparent ainsi pour une éventuelle automatisation. Par conséquent, le couplage des approches par planification avec les approches sémantiques constitue une réponse adéquate quant à l'exigence incessante en termes d'automatisation de la tâche de composition de services dans un environnement ambiant.

Adaptation : Comme nous l'avons souligné dans la **section 1.4.4 du chapitre 1**, l'auto-adaptation d'un système reflète sa capacité à gérer seul les changements de contexte afin de mieux s'adapter aux nouvelles situations. L'auto-adaptation demeure une nécessité absolue notamment pour les systèmes de composition de services dans des environnements ambiants. En effet, un service composite qui n'est pas adaptable, ne sera ni exploitable ni réutilisable lors d'un changement de contexte, par conséquent son intérêt sera assurément réduit. Afin qu'il soit auto-adaptable, un système de composition de services doit nécessairement intégrer des mécanismes d'adaptation au contexte. La sélection de services, le tissage d'aspects et l'apprentissage pour l'adaptation sont les trois principaux mécanismes d'adaptation au contexte. La sélection de services et le tissage d'aspects permettent une adaptation réactive tandis que l'apprentissage procure une adaptation proactive au système. L'adaptation par tissage d'aspects est adéquate pour la construction des applications à base de composants dans des domaines assez fermés et contrôlés. La raison pour laquelle ce type d'adaptation est adopté par les approches de composition structurelle. Pour le cas de notre thèse qui vise la construction d'application à base de services, l'adaptation par sélection de services semble la plus adéquate. Cependant, une telle adaptation est réalisée uniquement d'une manière réactive. Il est alors nécessaire de la renforcer par une adaptation proactive. Ainsi, le couplage de la sélection de services avec un mécanisme d'apprentissage procure une adaptation à la fois réactive et proactive pour un système de composition de services.

Complétude : Comme nous l'avons souligné dans la **section 2.6.3 du chapitre 2**, la problématique de la composition de services se situe au cœur des différents modèles orientés service. Elle représente le noyau auquel sont greffés les autres modules fonctionnels d'un

système. Ces modules représentent, en quelque sorte, les aspects en relation directe avec le problème de la composition de services notamment la découverte, la sélection, le monitoring et l'invocation des services. Bien que ces aspects puissent êtres réalisés indépendamment l'un de l'autre, leur mise en relation autour de l'approche de composition de services contribue fortement à la gestion complète d'un service composite, et ce, depuis sa création jusqu'à son exécution. En effet, dans une phase initiale, une approche de composition de services peut faire appel au module de découverte de services lors de la génération d'un service composite. Dans une phase intermédiaire, cette approche peut faire appel au module de sélection de services pour le choix des services adaptés au contexte d'utilisation. Enfin, dans une phase finale, l'approche de composition de services peut faire appel aux modules de monitoring et d'invocation pour exécuter concrètement le service composite. Par conséquence, la complétude d'une approche de composition de services dépend de sa capacité à tenir compte de cet ensemble d'aspects en relation directe avec le problème de la composition de services. Bien qu'elles soient différentes en termes de leur degré de complétude, les approches intergicielles offrent bien une meilleure capacité à résoudre un certain nombre de problèmes en relation avec la composition de services, entre autres, la découverte et l'invocation des services. De plus, ce type d'approches donne une vision globale sur le fonctionnement d'un système de composition de services. Cette vision permet ainsi d'adresser une grande partie des exigences des systèmes intelligents ambiants.

Compte tenu de l'analyse précédente, nous considérons qu'un système de composition de services dans un environnement ambiants peut être réalisé dans le cadre des approches intergicielles, et ce, afin d'assurer une visibilité meilleure, nette et complète du système. Toutefois, l'approche intergicielles doit être couplée avec les approches sémantiques et par planification afin de procurer un haut degré d'automatisation pour le système. Le résultat sera donc une hybridation des trois approches de composition de services. L'approche résultante doit être aussi dotée de mécanismes de sélection de services et d'apprentissage pour assurer une adaptation réactive et proactive pour le système de composition de services. C'est dans cette optique que nous allons analyser et présenter en détail, dans le **chapitre 4** suivant, le fonctionnement de certaines approches intergicielles de composition de services.

CHAPITRE 4

Etude et analyse des approches intergicielles de composition de services

4.1. Introduction

Plusieurs travaux se sont menés, par différentes communautés, aussi bien dans la recherche académique que dans l'industrie, sur la problématique de la composition de services. Ces travaux ont donné naissance à plusieurs orientations, qui constituent aujourd'hui autant d'approches pour la composition de services. A l'inverse des autres approches qui se focalisent uniquement sur certains aspects particuliers de la composition de services, tels que la sémantique des services et l'automatisation de la création des services composites, les approches integicielles se distinguent par leur vision globale sur le fonctionnement d'un système de composition de services. En effet, ces approches se basent sur un middleware (intergiciel) qui fournit des mécanismes permettant de résoudre un certain nombre de problèmes en relation directe avec la composition de services, notamment la découverte, la sélection, le monitoring et l'invocation des services. Ces mécanismes permettent ainsi aux applications de devenir adaptables, reconfigurables et tolérantes aux pannes. Grâce à cette particularité, les approches intergicielles de composition de services sont les mieux placées afin de répondre à une grande partie des exigences des systèmes intelligents ambiants.

Les middlewares de composition de services se caractérisent par deux points essentiels. Le premier point concerne le registre qui contient des informations sur les services disponibles, tandis que le deuxième point concerne le moteur qui compose et exécute réellement ces différents services. Bien qu'il existe de nombreux types de ces middlewares pour la composition de services, nous nous limitons uniquement, dans ce chapitre, à certains middlewares qui supportent la composition de services en informatique ubiquitaire et ambiante. Au minimum, trois fonctionnalités sont requises pour ce genre de middleware. Il s'agit d'abord de fournir une méthodologie claire et compréhensible pour la construction d'un service composite. Ensuite, l'approche d'évaluation adoptée par le middleware ne devrait pas être limitée uniquement aux ordinateurs de bureau mais elle doit tenir compte d'une large gamme de dispositifs distribués dans un environnement ambiant. En outre, le middleware doit se reposer sur une infrastructure de communication permettant l'accès aux différents dispositifs physiques qui supportent plusieurs standards de communication filaire et sans fil. D'autres fonctionnalités, telles que la découverte, la sélection et le monitoring de services, peuvent être facultatives, mais leur prise en compte améliore considérablement l'exhaustivité du middleware et la qualité qu'il engendre.

Compte tenu de ces spécifications et à la lumière de notre étude des différents travaux récents portant sur la composition de services, nous avons identifié onze principaux middlewares permettant la composition de services dans des environnements ambiants à savoir: COCOA-PERSE *(COnversation-based service Composition in pervAsive computing*

environments - Pervasive Semantic-aware Middleware), PEIS-Ecologie (*Physically Embedded Intelligent System),* URC-SURF *(Ubiquitous Robot Compagnon- Service-oriented Ubiquitous Robotic Framework),* MySIM (*Spontaneous Service Integration for Pervasive Environment)*, MEDUSA*, MUSIC (*Middleware Support for Self-Adaptation in Ubiquitous and Service-Oriented Environments),* SeSCo (*Seamless Service Composition)*, SeGSeC (*Semantic Graph-Based Service Composition*), URS (*Ubiquitous Robotic Space)*, VRESCo (*Vienna Runtime Environment for Service-oriented Computing*), *et* WComp (*Middleware for Ubiquitous Computing).*

L'objectif global de ce chapitre consiste alors à étudier et analyser ces différents middlewares selon les dix modèles orientés services cités dans la **section 2.6.2 du chapitre 2** à savoir : l'architecture générale, le modèle des connaissances, le modèle d'évaluation de la QoS et du contexte, le modèle de *matching* des services, le modèle de sélection des services, le modèle de découverte des services, le modèle de composition des services, le modèle de monitoring des services, le modèle d'évaluation, et le modèle de communication. Ces différents modèles visent la réalisation, dans un cadre orienté service, des exigences des systèmes intelligents ambiants détaillées dans la **section 1.4 du chapitre 1** à savoir: l'intelligence, la sensibilité au contexte, l'auto-adaptation, l'intéropérabilité, la transparence et le passage à l'échelle. Dans ce contexte de réalisation, le modèle de composition de services représente le noyau du système intelligent auquel sont greffés les autres modèles orientés services comme modules fonctionnels de ce système.

Dans ce chapitre, nous étudions et analysons d'abord les différents middlewares cités précédemment suivants les dix modèles orientés services. Ensuite, nous comparons et positionnons chaque middleware étudié par rapport aux autres en termes de la prise en compte de certains défis et exigences des environnements ambiants. Enfin, nous terminons ce chapitre par une conclusion générale portant sur la partie de l'état de l'art.

4.2. Les middlewares de composition de services

4.2.1. COCOA-PERSE

4.2.1.1. Architecture générale

Dans **[252]**, les auteurs présentent le Framework COCOA (*COnversation-based service Composition in pervAsive computing environments*), comme solution pour la composition de service en informatique ubiquitaire et *pervasive* basée sur l'intégration des conversations avec support de la *QoS*. COCOA fournit COCOA-L, un langage basé sur OWL-S pour la spécification sémantique de la *QoS* et de la conversation des services et des tâches. En outre, COCOA fournit deux mécanismes: COCOA-SD pour une découverte des services avec prise en compte de la *QoS* et COCOA-CI pour l'intégration des conversations des services pour la réalisation de la conversation de la tâche de l'utilisateur en tenant compte aussi de la *QoS*.

Comme extension à COCOA, les auteurs présentent dans **[93]** un Framework plus générique, appelé PERSE (*Pervasive Semantic-aware Middleware*), qui intègre toutes les fonctionnalités offertes dans COCOA, complétées par la gestion multi-réseaux et multi-protocoles fournie par le middleware MUSDAC **[253]**. Comme le montre la **figure 4.1**,

PERSE est composé de deux couches principales: la couche middleware de communication et la couche middleware sémantique orientée services (SOM). La couche sémantique SOM implémente des fonctionnalités, telles que, la publication, la localisation (découverte), le *matching* et la composition des services développés dans l'approche COCOA. Le middleware de communication supporte aussi le multi-réseaux et le multi-protocoles lors de la découverte et l'accès aux services.

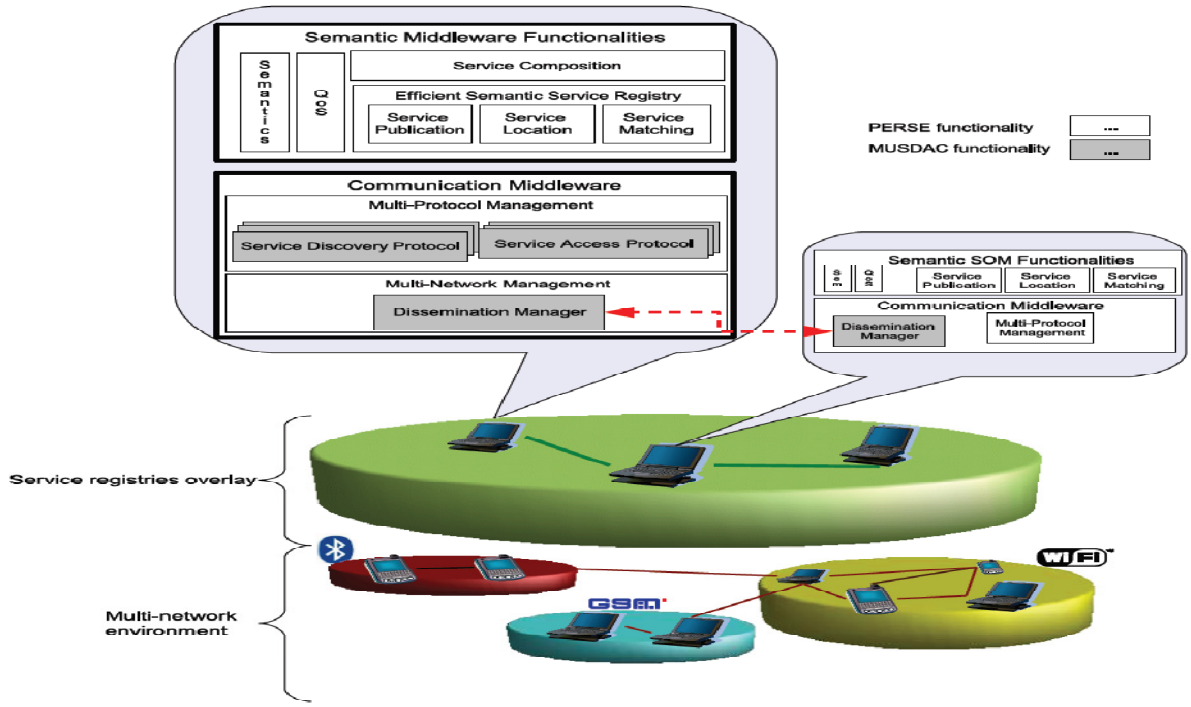


Figure 4.1 : Le Middleware PERSE **[93]**.

4.2.1.2. Modèle des connaissances

COCOA-L est un langage basé sur OWL-S qui permet la spécification de trois aspects principaux: (1) les capacités fonctionnelles des services et des tâches annoncés et demandés; (2) les conversations des services et des tâches pour la modélisation de leur comportement, et (3) les propriétés de la QoS des services et des tâches. COCOA-L étend OWL-S avec les principaux éléments conceptuels suivants:

- ***Capacité***: une capacité caractérise une fonctionnalité qui pourrait être annoncée ou demandée par un service ou une tâche. Elle est réalisée par l'invocation d'un ensemble d'opérations.
- ***Opération****:* il s'agit d'une séquence de messages échangés entre un client et un fournisseur de services, par exemple, une opération WSDL.
- ***Capacité demandée***: il s'agit d'une capacité qui fournit un ensemble d'entrées, et qui requière une catégorie, un ensemble de sorties et des propriétés de qualité de service.
- ***Capacité annoncée***: il s'agit d'une capacité qui requière un ensemble d'entrées, et qui fournit une catégorie, un ensemble de sorties et des propriétés de la qualité de service.
- ***Conversation***: une conversation représente la coordination d'un ensemble de capacités des services et des tâches par des structures de contrôle OWL-S, par exemple, des séquences, des structures parallèles, des structures de choix, etc. En COCOA-L, la conversation d'une

tâche de l'utilisateur comprend les capacités demandées par cette tâche, tandis que la conversation d'un service comprend les capacités annoncées par ce service.

- ***Dépendances de contrôle***: ces dépendances sont dues à la structure de la conversation. Plus précisément, deux capacités C1 et C2 sont dites avoir une dépendance de contrôle si C1 est antérieure à C2 dans la conversation, et afin d'assurer une consommation valide d'un service ou d'une tâche, C1 doit être effectuée avant C2 lorsque le service (ou la tâche) est exécuté. Cependant, lors de la réalisation de la tâche de l'utilisateur, l'entrelacement des conversations de multiples services reste possible tant que les dépendances de contrôle de ces services sont satisfaites.
- ***Dépendance de données***: il existe une dépendance de données entre deux capacités C1 et C2 si les données produites par C1 doivent être consommées uniquement par C2. Lorsqu'une dépendance de données est spécifiée dans la conversation d'une tâche de l'utilisateur, cela signifie que les deux capacités correspondantes doivent être fournies par le même service. Si une dépendance de données est spécifiée dans une conversation d'un service, cela signifie que les capacités correspondantes doivent être effectuées en séquence et sans entrelacement avec d'autres capacités en dehors de la conversation de ce service.
- ***La spécification du flux de données***: un flux de données spécifie les données de sortie produites par une capacité qui seront consommées par une autre capacité. Les capacités concernées doivent être compatibles en termes d'entrées/sorties qui seront échangées selon la spécification du flux de données.

COCOA-L permet également la spécification des dimensions quantitatives et qualitatives de la *QoS*. Les dimensions quantitatives mesurent des attributs spécifiques quantifiables d'un service tandis que les dimensions qualitatives décrivent le comportement du service. Ces dimensions sont décrites dans COCOA-L par des références à des concepts ontologiques. Les dimensions de la *QoS* représentent les attributs de qualité fournis par les services. Selon leur nature, ces attributs sont classés en plusieurs catégories de qualité de service. Une propriété de la *QoS* est décrite en fonction des dimensions de qualité de service et exprimée comme une expression booléenne en utilisant certains opérateurs comme: *et*, *ou*, *non*, *égalité*, *inférieur à*, etc. Certains de ces opérateurs sont utilisés pour comparer des propriétés qualitatives, tandis que d'autres sont utilisés pour comparer des propriétés quantitatives. Les opérateurs *et*, *ou* et *non* sont utilisés pour définir des propriétés composites.

4.2.1.3. Modèle d'évaluation de la *QoS* et du contexte

Une tâche d'un utilisateur requière deux types de propriétés de *QoS*: propriétés de *QoS* spécifiées au niveau des capacités exprimant des exigences de qualité de service locales et des propriétés de *QoS* spécifiées pour toute tâche exprimant des exigences de qualité de service globales. Les exigences locales de qualité de service doivent être satisfaites par les différentes capacités annoncées par les services, tandis que les exigences globales doivent être satisfaites par la composition de service qui en résulte. Afin d'évaluer la qualité de service d'une composition de service, deux estimations pour chaque dimension *QoS* sont fournies: (1) une estimation probabiliste basée sur l'historique, et (2) une estimation pessimiste. La première correspond à une estimation moyenne, tandis que la seconde correspond à une estimation au pire des cas. L'utilisation des deux estimations précédentes dépend des exigences de la tâche

de l'utilisateur (déterministe ou probabiliste) spécifiées dans la requête de l'utilisateur. L'importance relative est utilisée pour caractériser les préférences des utilisateurs parmi les différentes dimensions de qualité de service et la criticité des ressources des dispositifs hébergeant les services. L'approche proposée pour l'évaluation de la *QoS* des tâches composées pour les utilisateurs extrait d'abord les formules de la *QoS* correspondant à chaque dimension de la *QoS*. Ces formules sont extraites à l'avance et stockées avec la description de la tâche. Puis, lors de la composition, à chaque fois qu'un élément est composé, ces formules sont utilisées pour vérifier le respect des exigences de qualité de service de la tâche. L'approche d'évaluation utilise le modèle mathématique basé sur des règles de réduction, proposé dans **[254]**, pour extraire la formule de chaque dimension de la *QoS* en fonction de la structure de l'automate de la tâche. Toutefois, COCOA utilise uniquement les règles de réduction portant sur les structures séquentielles, conditionnelles, et avec boucles.

4.2.1.4. Modèle de *matching* de services

Le modèle proposé utilise la relation de *matching Match (Adv, Req)* qui fait correspondre une capacité annoncée *Adv* à une capacité demandée *Req*. Cette relation étend la relation définie dans **[255]** avec le *matching* des propriétés de la *QoS*. Plus précisément, la relation *Match* est définie en utilisant la fonction *distance (concept1, Concept2)* qui donne la distance sémantique entre les deux concepts, *concept1* et *concept2*, selon la classification donnée par l'ontologie à laquelle les deux concepts appartiennent. Si *concept1* ne subsume pas (subsume signifie englober et exprime le fait d'intégrer quelque chose dans une catégorie plus générale) *concept2* dans l'ontologie à laquelle ils appartiennent, la distance entre les deux concepts n'a pas une valeur numérique, c.-à-d *distance (concept1, concept2) = NULL*. Si *concept1* subsume *concept2* alors la distance prend comme valeur le nombre de niveaux qui séparent *concept1* de *concept2* dans la hiérarchie de l'ontologie à laquelle ils appartiennent. Dans cette relation, le cas où *concept1* est subsumé par *concept2* est considéré comme un déséquilibre et la valeur *NULL* est alors affectée à la relation *distance (concept1, concept2)* parce que cela implique que le client peut recevoir une capacité annoncée plus spécifique que celle demandée, ce qui peut conduire à un dysfonctionnement de la capacité annoncée. En outre, pour la réalisation automatique de la tâche de l'utilisateur, seulement les capacités qui sont équivalentes ou plus génériques que les capacités demandées sont sélectionnées, évitant ainsi le risque de dysfonctionnement des capacités. La relation *Match (Adv, Req)* est satisfaite si et seulement si: (1) toutes les entrées requises par *Adv matchent* avec les entrées fournies par *Req*; (2) toutes les sorties requises par *Req matchent* avec les sorties fournies par *Adv*; (3) la catégorie requise par *Req matche* avec la catégorie fournie par *Adv*, et (4) toutes les propriétés requises par *Req matchent* avec les propriétés fournies par *Adv*.

4.2.1.5. Modèle de sélection de services

La sélection des services est effectuée par COCOA-SD. Elle est basée sur les dépendances de données et de contrôle qui sont inhérentes à la spécification des conversations des services. Par exemple, un service qui fournit une capacité sémantiquement équivalente à l'une des capacités demandées par la tâche de l'utilisateur ne pourrait pas être utile pour la composition, si cette capacité possède des dépendances de données ou de contrôle avec d'autres capacités qui ne sont pas demandés par cette même tâche de l'utilisateur. Cette sélection est effectuée en

utilisant une expression régulière correspondant au langage généré par l'automate de la tâche. Pour chaque terme de cette expression régulière, qui correspond à une capacité de la description de la tâche, le quantificateur *?* est introduit pour indiquer qu'il y a 0 ou 1 occurrence de ce terme. Notons par L le langage généré par l'expression régulière extraite et par $L_1, L_2, \ldots, L_n$ les langues générés par les automates des services de présélectionnés $S_1, S_2, \ldots, S_n$, respectivement. COCOA-SD sélectionne tous les services S_i tel que $L \cap L_i \neq \emptyset$. Ceci permet la sélection des services qui répondent aux dépendances de contrôle de la tâche de l'utilisateur en permettant l'entrelacement potentiel de leurs conversations. En outre, si une dépendance de données est spécifiée entre deux capacités de la tâche de l'utilisateur, seuls les services qui offrent à la fois ces deux capacités dans leur conversation sont conservés parmi les services précédemment sélectionnés. La sélection des services est également basée sur les spécifications de la *QoS*. En particulier, si des exigences locales de la *QoS* sont spécifiées dans certaines capacités de la tâche de l'utilisateur, les capacités des services qui ne répondent pas à ces exigences ne sont pas sélectionnées pour la composition.

4.2.1.6. Modèle de découverte de services

La découverte des services permet de trouver dans l'environnement *pervasif*, au moment et au lieu précis, les capacités annoncées qui *matchent* les capacités demandées pour la réalisation de la tâche de l'utilisateur. La découverte de services est effectuée par COCOA-SD qui se base sur le *matching* et la sélection des services décrits dans les deux sections précédentes.

4.2.1.7. Modèle de composition de services

Les conversations des services et des tâches sont modélisées en utilisant les automates à états finis (FSA). Des règles de *mapping* sont définies pour la translation d'un modèle de processus OWL-S à un automate à états finis. Dans ce modèle, les symboles de l'automate correspondent aux capacités décrites à l'aide de COCOA-L. L'état initial correspond au début de la conversation, et les états finaux correspondent à la fin d'une interaction client/service. Chaque structure de contrôle impliquée dans une conversation est transformée à un automate à l'aide des règles de *mapping* définies. Puis, ces automates sont reliés entre eux afin de construire un automate global. Une fois la découverte sémantique des services est achevée par COCOA-SD, la prochaine étape pour la composition dynamique de la tâche de l'utilisateur est l'intégration des conversations des services sélectionnés à l'aide de COCOA-CI qui se base sur les automates à état finis associés. COCOA-CI intègre d'abord tous les automates des services sélectionnés dans un automate global. Ce dernier contient un nouvel état initial relié par des transitions vides avec les états initiaux de tous les automates sélectionnés. L'automate global contient également d'autres transitions vides qui relient son état initial aux états finaux de chaque automate sélectionné. Ensuite, COCOA-CI analyse chaque état de l'automate de la tâche en commençant par son état initial, et en suivant ses transitions. Simultanément, une analyse de l'automate global est effectuée afin de trouver pour chaque état de l'automate de la tâche un état de l'automate global qui peut le simuler. Un état de l'automate de la tâche est dit simulé par un état de l'automate global si pour chaque symbole entrant (symboles entrants d'un état correspondent aux étiquettes des transitions suivantes de cet état) de l'état de l'automate de la tâche, il y a au moins un symbole entrant sémantiquement équivalent de

l'état de l'automate global. COCOA-CI permet de trouver des compositions de services avec entrelacement possible des conversations des services impliqués. En effet, cela se fait par la gestion des sessions de service. Une session de service caractérise l'état d'exécution d'une conversation de service. Une session est ouverte quand une conversation de service démarre et se termine lorsque cette conversation se termine. Une condition importante de la gestion des sessions est que chaque session qui a été ouverte doit être fermée, c'est à dire qu'il faut arriver à un état final de l'automate du service. Durant le processus de composition, des chemins différents dans l'automate global, qui représentent des compositions intermédiaires, sont étudiés. Certains de ces chemins seront rejetés lors de la composition tandis que d'autres seront conservés (par exemple, un chemin sera rejeté si ce dernier contient un service dans lequel une session a été ouverte, mais n'a jamais été fermée). En plus de vérifier pour chaque état l'équivalence entre les capacités entrantes, une vérification de la conformité aux contraintes de la *QoS* de la tâche de l'utilisateur est effectuée. Ceci est fait en utilisant les formules de la *QoS* qui ont été extraites de la structure de l'automate de la tâche (formule de *QoS* pour chaque dimension de la *QoS*). COCOA-CI donne un ensemble de sous-automates de l'automate global qui sont conformes à la structure de l'automate de la tâche. Chacun de ces automates est une composition de services qui est conforme à la conversation de la tâche de l'utilisateur. Une fois l'ensemble des compositions possibles est donné, une dernière étape consiste à choisir la meilleure composition sur la base de la *QoS* fournie.

4.2.1.8. Modèle d'évaluation

Pour illustrer le genre de situations dans lesquelles leur approche est applicable, les auteurs dans **[252]** présentent le scénario de l'application *e-movie* décrite comme suit: dans une gare, une personne attend le départ de son train dans une salle où un certain nombre de services numériques sont disponibles, parmi lesquels un service de *streaming* utilisé pour diffuser des ressources numériques sur les appareils portables des utilisateurs, et un grand écran plat qui diffuse en continu des informations. La personne décide de regarder un film en utilisant l'application *e-movie* qu'il possède sur son PDA, à laquelle il spécifie comme entrée le titre du film qu'il veut regarder. Cette application est en mesure de découvrir les serveurs vidéo ainsi que les dispositifs d'affichage disponibles qui sont à la portée de cet utilisateur. Elle permet également de choisir le dispositif le plus approprié. Plus précisément, si un écran plus grand que l'écran PDA de l'utilisateur se trouve dans sa portée, et si le contexte de cet utilisateur le permet (par exemple, personne d'autre ne se trouve dans la même salle), cette application affiche le film sur cet écran. En outre, si le contexte de l'utilisateur change (par exemple, l'utilisateur quitte la salle ou une personne entre dans la salle), l'application est capable de transférer le flux vidéo vers le PDA de l'utilisateur.

4.2.1.9. Modèle de communication

PERSE intègre les fonctionnalités d'accès au service fournies par MUSDAC. Ce dernier prend en charge l'accès des clients à des services hébergés dans des réseaux distants et suppose que les clients et les services utilisent SOAP comme protocole d'accès (par exemple, UPnP et les services Web). Dans ce cas, la traduction des messages est simplifiée, car seuls les en-têtes des messages doivent être modifiés (pour la gestion des niveaux de routage de l'application), tandis que le contenu des messages d'accès reste le même. L'accès à un service

à distance via PERSE est réalisé grâce à la création d'un canal de communication. La création de ce canal reste transparente pour le client et se fait lors du premier accès au service distant. Lorsque le client initie une interaction avec un service, il utilise une adresse locale qui a été ajoutée à la description du service par *PERSE Service* au lieu de l'adresse du service cible. Le canal de communication est composé de l'adresse du client, l'adresse locale (fournie par *PERSE Service*) et la liste de diffusion ; c'est-à-dire la liste de tous les ponts entre le client et le service cible. Une fois créés, les messages du client sont traduits (par exemple, changer l'en-tête du message SOAP) et encapsulés dans un message envoyé sur le canal de communication. Chaque gestionnaire de diffusion qui reçoit un message d'accès vérifie l'identificateur unique du canal de communication pour ce message et le retransmet jusqu'à ce qu'il atteigne le service cible. Le résultat est renvoyé d'une manière similaire au client.

4.2.2. PEIS-Ecology

4.2.2.1. Architecture générale

Le concept de PEIS-Ecologie (*Physically Embedded Intelligent System*) **[256, 160, 161]** est une approche tout à fait nouvelle pour l'insertion des technologies robotiques dans les environnements quotidiens en associant les domaines de l'intelligence ambiante et de la robotique autonome. L'approche PEIS-Ecologie met l'accent sur le fait que nos environnements incorporent de plus en plus de dispositifs embarqués et d'objets taggués. Par conséquent, les fonctionnalités robotiques avancées ne sont plus réalisées par le développement de robots très sophistiqués mais grâce à la coopération de nombreux composants robotiques simples.

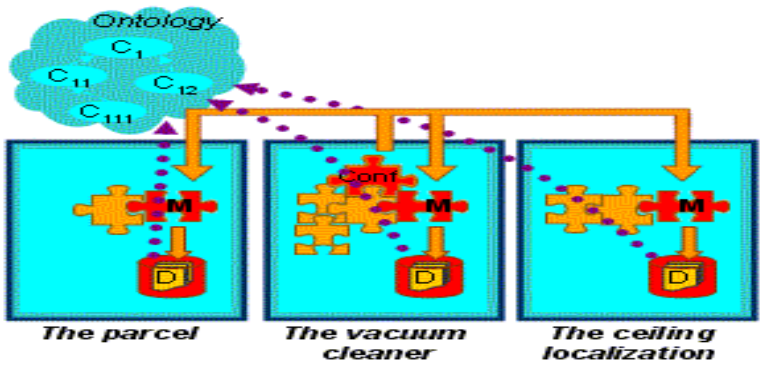


Figure 4.2 : Les composants de PEIS-Ecologie **[160]**.

La **figure 4.2** illustre l'approche adoptée dans le Framework PEIS-Ecologie. Chaque composant *PEIS* dispose d'un répertoire local de descriptions *D* et d'un composant spécial *M* qui peut accéder à ces descriptions et les annoncer pour le reste de l'écologie. Certain composants *PEIS* peuvent être équipés d'un composant particulier, appelé *configurateur*, désigné par *Conf*, qui est capable de récupérer des descriptions et calculer une configuration sur la base des informations contenues dans ces descriptions. Le configurateur s'occupe également du déploiement et du monitoring de la configuration générée. Pour la partie monitoring, le configurateur s'inscrit aux signaux d'erreur provenant des *PEIS* connectés, et re-déclenche l'algorithme de configuration si un composant PEIS de la configuration tombe en panne pour une raison quelconque.

Il est à noter que plusieurs configurateurs peuvent exister dans l'écologie et certains composants PEIS peuvent ne pas inclure un configurateur. Chaque fois qu'un PEIS, qui ne possède pas un configurateur, a besoin de générer une configuration, il sollicite le service d'un configurateur disponible dans d'autres *PEIS*. L'aspect hétérogénéité dans une écologie est géré à l'aide d'une ontologie. Cette dernière permet de décrire les fonctionnalités offertes par chaque composant *PEIS* de l'écologie et les données sur lesquelles ces composants opèrent, et de définir la notion de compatibilité utilisée par le mécanisme de découverte. L'approche PEIS-Ecologie intègre la dimension humaine afin de valider son utilité et son acceptabilité par les utilisateurs. La solution proposée au problème de l'interfaçage d'un PEIS-écologie avec les utilisateurs est basée sur deux nouveaux concepts **[257]**: point d'interface commune et expressions basées sur la sémantique. Le premier concept recueille et synthétise les informations pertinentes sur l'état du PEIS-Ecologie, et fait en sorte que l'utilisateur perçoit cette écologie comme un seul système. Le deuxième concept offre une méthode uniforme pour représenter l'état de chaque composant *PEIS* dans l'écologie par un degré de satisfaction, et de transmettre cette information à l'utilisateur par une expression compréhensible.

4.2.2.2. Modèle des connaissances

Le concept de PEIS-écologie s'appuie sur les ingrédients suivants:

- Le *PEIS* qui est un ensemble de composants logiciels interconnectés résidant dans une même entité physique. Chaque composant peut inclure des liens vers des capteurs et des actionneurs, ainsi que des ports d'entrée/sortie qui le relient à d'autres composants qui se trouvent soit dans le même PEIS ou dans d'autres PEIS. Chaque robot dans l'environnement est représenté par un PEIS comme une notion uniforme. Tous les PEIS sont reliés par un modèle de communication uniforme, ce qui permet l'échange d'informations entre les différents composants PEIS. Tous les PEIS dans une écologie peuvent coopérer via un modèle de coopération uniforme, basé sur la notion de liaisons entre les composants fonctionnels: chaque PEIS participant peut utiliser les fonctionnalités d'un autre PEIS de l'écologie afin de compléter ses propres fonctionnalités.
- Le *PEIS-Ecologie* qui est une collection de PEIS interconnectés et embarqués dans le même environnement physique nommé écologie. Similaire à un écosystème naturel, *PEIS-Ecologie* est caractérisé par: (i) la relation entre les entités existantes (PEIS) et l'environnement (grâce à des capteurs et des actionneurs), (ii) la relation entre ces entités (par l'intermédiaire du modèle de coopération), et (iii) l'idée qu'un comportement complexe du système émerge de l'interaction de plusieurs unités simples. En outre, une *PEIS-Ecologie* est destiné à être hétérogène en intégrant des espèces différentes d'entités en interaction symbiotique. L'homme peut constituer l'une de ces espèces qui participent à l'écologie, et peut même interagir avec les autres PEIS de cette écologie.
- La *Configuration d'une PEIS-Ecologie* qui est l'ensemble des connexions entre les composants à l'intérieur et entre les PEIS de l'écologie. Il est à noter que toutes les connexions entre les PEIS sont supportées par un middleware commun. Surtout, une même écologie peut être configurée de différentes façons en fonction du contexte d'utilisation, tels que, la tâche en cours, l'état de l'environnement et les ressources disponibles.

4.2.2.3. Modèle de *matching* de services

Dans PEIS-Ecologie, une approche de *matching* est utilisée afin de relier les informations perceptuelles acquises du monde physique sur les propriétés d'un objet, et les informations sur cet objet (le monde numérique) qui sont fournies par l'objet lui-même. Cette approche est basée sur une extension de la notion de perception d'ancrage (*perceptual anchoring*) **[258]**, qui est le processus de connexion, à l'intérieur d'un système intelligent, des symboles utilisés pour désigner un objet (par exemple, la parcelle box-22) et les percepts issus du même objets (par exemple, un blob vert dans l'image de la caméra). Considérons par exemple la situation où un robot est entrain de percevoir un PEIS-box vert (**Figure 4.3**), qui est étiqueté comme box-22. Comment ce robot peut décider qu'afin de connaître le poids de cette boîte, il doit lire la propriété de poids à partir du PEIS identifié par l'ID = Peis4?

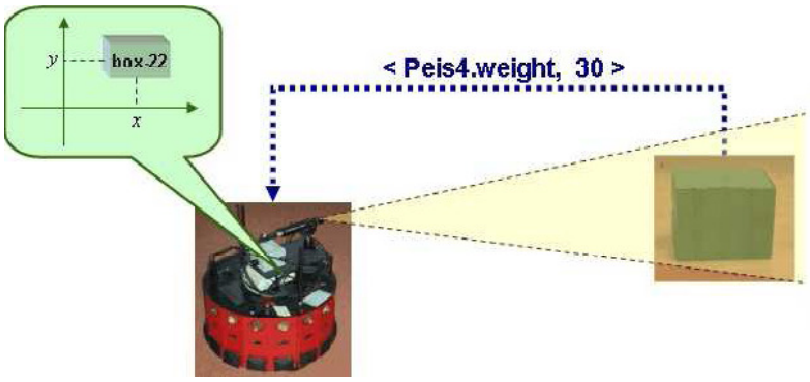


Figure 4.3 : Acquisition des informations sur un objet à la fois par la perception et par la communication numérique **[160]**.

Un mécanisme similaire à la primitive *Find* du Framework d'ancrage (*anchoring framework*) **[258]** est utilisé dans cette approche. Chaque PEIS publie sa représentation physique sous forme d'un tuple dans le *tuple-space.* Le robot interroge le *tuple-space* pour tous les *tuples* de chaque PEIS dans l'écologie. Il essaie ensuite de faire correspondre (matcher) ces *tuples* aux propriétés perçues de la parcelle en face de lui, par exemple, en forme de boîte, vert, et d'une taille donnée. Dans l'exemple illustré par la **figure 4.3**, le *matching* réussit pour Peis4. Une fois cela est fait, le robot peut demander des propriétés supplémentaires à Peis4 (par exemple, son poids) et combiner ces propriétés avec les valeurs observées. En général, les informations perceptives et symboliques sur le même objet provenant de plusieurs PEIS peuvent être *matchées* et combinées de la même manière. Ce mécanisme est exploré en détail dans **[259]**.

4.2.2.4. Modèle de découverte de services

Le mécanisme de découverte permet à chaque PEIS de trouver d'autres PEIS qui peuvent fournir des fonctionnalités compatibles avec ses propres besoins. Pour traiter l'aspect hétérogénéité des PEIS dans une écologie, une ontologie est utilisée afin de décrire les fonctionnalités offertes par chaque composant PEIS et les données sur lesquelles ils opèrent. L'ontologie permet également de définir la notion de compatibilité utilisée par le mécanisme de découverte. Ainsi, la compatibilité entre les fonctionnalités annoncées et celles requises est décidée en utilisant une ontologie partagée entre les différents PEIS de l'écologie. Par exemple, quand un composant PEIS requiert une fonctionnalité, il cherche tout simplement un tuple annonçant une fonctionnalité compatible dans n'importe quel autre composant PEIS: si

un composant est trouvé, alors ce composant sera réservé et un abonnement lui sera créé. Grâce au PEIS-kernel, à tout moment, chaque composant *PEIS* peut détecter la présence d'autres composants et négocie avec eux l'utilisation de leurs fonctionnalités. PEIS-kernel peut également gérer le fait qu'un PEIS peut dynamiquement rejoindre et quitter l'écologie.

4.2.2.5. Modèle de composition de services

Dans l'approche PEIS-Ecologie, la tâche et la configuration ne sont pas prédéfinies. Donc, l'écologie devrait être en mesure de s'auto-configurer dynamiquement afin de s'adapter à la tâche en cours et à la situation actuelle où la PEIS-Ecologie est intrinsèquement dynamique (un PEIS peut rejoindre et quitter l'écologie à tout moment). La **figure 4.4** illustre le problème d'auto-configuration dans une PEIS-Ecologie. En général, le processus de configuration fonctionne en trois phases: évaluer l'état actuel de l'écologie (introspection), par exemple, localiser les fonctionnalités disponibles; générer une configuration appropriée pour la tâche cible; et instancier cette configuration sur l'écologie en établissant les paramètres et les inscriptions nécessaires.

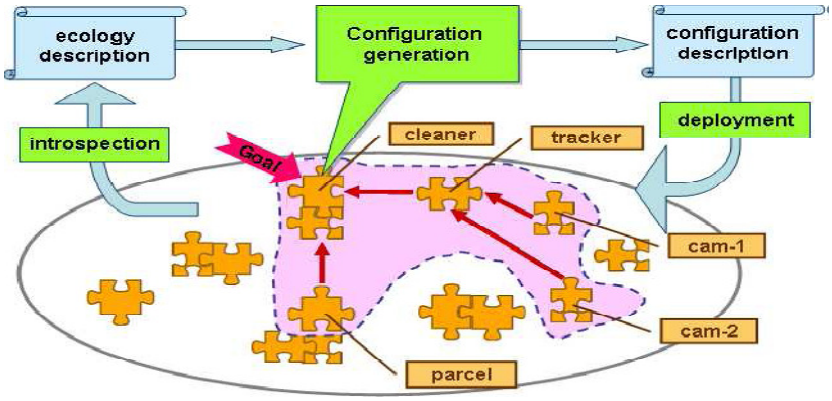


Figure 4.4 : L'auto-configuration d'une PEIS-Ecologie **[160]**.

Le composant *configurateur* est chargé de la récupération des descriptions et le calcul d'une configuration sur la base des informations stockées dans ces descriptions. Chaque fois qu'un PEIS, qui ne possède pas un *configurateur*, a besoin de générer une configuration, il sollicite le service d'un *configurateur* disponible dans d'autres PEIS. Le *configurateur* est implémenté en utilisant deux approches complémentaires: une approche centralisée basée sur le plan (*plan-based, centralized approach*) **[260]**, et une approche distribuée réactive (*reactive, distributed approach*) **[261]**. Dans la première approche, un planificateur hiérarchique global est utilisé pour générer une configuration avec coût minimum pour une tâche donnée. Dans la seconde approche, le *configurateur* crée une configuration partielle, et suppose que les PEIS connectés sont capables d'étendre récursivement cette configuration si c'est nécessaire. Lorsqu'un PEIS tombe en panne, le *configurateur* reçoit un signal de défaillance et tente une configuration locale différente. Les deux approches ont les points forts et les faiblesses des approches basées sur le plan et des approches réactives. L'approche basée sur le plan garantie de trouver la configuration optimale si elle existe, mais elle ne peut pas faire facilement face aux changements dans l'écologie. L'approche réactive, quant à elle, est capable de s'adapter rapidement aux changements de l'état de l'écologie, mais elle pourrait générer des configurations non optimales et elle peut échouer à trouver une configuration, même si celle-ci existe.

4.2.2.6. Modèle de monitoring de services

L'approche proposée pour l'auto-configuration est basée sur les mécanismes d'annonce (*advertising*) et de monitoring pour faire face à l'aspect dynamique de l'écologie. Le mécanisme d'annonce permet à tous les PEIS de rejoindre dynamiquement l'écologie et de publier ses fonctionnalités pour les faire connaître à tous les autres PEIS, alors que le mécanisme de monitoring est en mesure de modifier une configuration dont des fonctionnalités ne sont plus disponibles. C'est le *configurateur* qui s'occupe du monitoring de la configuration générée. Pour se faire, le *configurateur* s'inscrit aux signaux d'erreur provenant des PEIS connectés, et re-déclenche l'algorithme de configuration si un composant PEIS de la configuration tombe en panne pour une raison quelconque. L'auto-adaptation a été laissée comme perspective pour des travaux futurs. Une certaine forme d'autoréparation est fournie comme une partie de l'approche de l'auto-configuration: si les signaux des PEIS dont les fonctionnalités qui sont utilisées dans la configuration actuelle ne sont plus disponibles, le *configurateur* tente alors de générer une configuration alternative.

4.2.2.7. Modèle d'évaluation

Afin de valider l'approche PEIS-Ecologie sur des scénarios réels, les auteurs dans **[160]** ont construit un testbed, appelé PEIS-Home, qui ressemble à un studio typiquement suédois (**Figure 4.5**). PEIS-Home est équipé d'une infrastructure de communication et d'un certain nombre de composants PEIS, tels que les caméras statiques, les robots mobiles, les nœuds capteurs, un réfrigérateur équipé de capteurs de gaz et d'un lecteur RFID, etc.

Figure 4.5 : Deux vues du *testbed* de PEIS-Ecologie **[160]**.

Deux exemples de scénarios sont typiquement illustrés: le premier est un aspirateur autonome dans une maison et le second est un robot de surveillance d'une maison à l'aide d'une sensation électronique (*e-nose*).

Scénario 1 (un aspirateur autonome dans une maison): un aspirateur autonome est un robot de nettoyage considéré comme un simple PEIS qui ne possède pas assez de ressources de détection et de raisonnement pour évaluer sa propre position dans la maison. Supposons que la maison est équipée d'un système de suivi par caméras (*tracking*), lui-même est un PEIS. Ensuite, ces deux PEIS sont combinés dans une simple PEIS-Ecologie dans laquelle le système de suivi fournit une fonctionnalité de localisation globale à la composante de navigation du robot de nettoyage, qui peut ainsi appliquer une stratégie intelligente pour le nettoyage. Supposons en outre que le robot nettoyeur rencontre une parcelle inattendue sur le sol. Le robot a besoin de connaitre le poids de cette parcelle afin de la repousser et nettoyer au

dessous. Par ailleurs, la meilleure façon d'accéder aux propriétés de la parcelle est de stocker ces propriétés dans la parcelle elle-même. Si la parcelle est équipée d'un *IC-tag*, elle peut agir comme un PEIS et communiquer ces informations (ex. poids) directement au robot nettoyeur. Dans cet exemple, la localisation globale est fournie au robot par le système de suivi à l'aide des caméras statiques. Maintenant, si le robot sort des champs de vision de ces caméras, l'écologie peut être alors reconfigurée pour laisser le robot nettoyeur utiliser son propre composant odométrique pour la localisation.

Scénario 2 (robot de surveillance d'une maison par une sensation électronique (*e-nose*)): Comme un exemple plus complexe, les auteurs considèrent un robot mobile qui devrait surveiller une maison à l'aide d'une sensation électronique (*e-nose*), par exemple, pour détecter la nourriture qui périt, fuites de gaz ou d'autres problèmes. Le robot doit détecter les odeurs anormales, aller vers la source, et classer l'odeur. Le *e-nose* doit être placé près de la source de l'odeur afin de classer cette odeur, mais la localisation de la source en suivant l'odeur est encore un problème non résolu. En outre, la classification des odeurs ne peut se faire de manière fiable que si le nombre de classes possibles est petit. La solution proposée par PEIS-Ecologie à ce problème serait la suivante. Tout d'abord, il est supposé que l'environnement contient un certain nombre de *e-nose* simples, petit, pas cher et placés à des endroits critiques, par exemple, à l'intérieur du réfrigérateur ou près de la cuisinière. Ces dispositifs simples peuvent détecter une concentration anormale de gaz, mais ils sont incapables de classer le type d'odeur. Il est également supposé que les objets dans l'environnement (par exemple, la nourriture dans le réfrigérateur) peuvent avoir des étiquettes RFID attaché, qui contiennent des informations sur l'objet lui-même. Quand un simple *e-nose* détecte une alarme, son emplacement est envoyé à un robot mobile équipé d'un système *e-nose* sophistiqué et coûteux. Le robot se déplace alors à cet endroit pour mieux sentir les odeurs des différents objets de cet endroit. Les informations stockées dans l'étiquette RFID de l'objet sont alors communiquées au robot. Ces informations fournissent un contexte afin de limiter le problème de classification. Cette solution a été étudiée et validée expérimentalement dans **[262]**.

4.2.2.8. Modèle de communication

Toutes les connexions entre les différents PEIS de l'écologie sont supportées par un middleware partagé appelé PEIS-kernel **[263]** qui gère l'hétérogénéité. Ce middleware fournit des primitives de communication uniformes ainsi que d'autres services, tels que la découverte des réseaux et le routage des messages entre PEIS. Une première version open-source de PEIS-kernel a été livrée en 2006 sous un ensemble de licences GNU, et elle est disponible sur le site Web du projet PEIS-Ecologie. PEIS-kernel implémente un modèle de communication basé sur un *tuple-space* distribué muni des opérations habituelles d'insertion et de lecture. En outre, il fournit des primitives basées sur l'abonnement et le désabonnement à des événements, par lesquelles un composant PEIS peut manifester son intérêt pour un tuple donné. Lorsqu'une opération d'insertion est effectuée, tous les abonnés sont informés. L'abonnement, la notification, et la distribution des tuples sont gérés par le PEIS-kernel d'une manière transparente aux composants PEIS. Cette approche hybride *tuples/événements* est de plus en plus utilisée dans l'informatique ubiquitaire et l'intelligence ambiante **[264, 265]**.

4.2.3. URC-SURF

4.2.3.1. Architecture générale

Dans **[163]**, les auteurs propose de réaliser leur vision d'un robot ubiquitaire compagnon URC (*Ubiquitous Robotic Companion*) par un Framework orienté services appelé SURF (*Service-oriented Ubiquitous Robotic Framework*) qui permet l'intégration automatique des robots dans des environnements ubiquitaires. Comme le montrent la **figure 4.6**, l'architecture de SURF est constituée de trois composantes majeures, qui sont SA (*SURF Agent*), DWS (*Device Web Service*) et EKR (*Environmental Knowledge Repository*). Le SA est un agent de planification intelligente et de demande (requête) sémantique des services Web. Cet agent joue un rôle majeur dans la procédure d'intégration automatique dans l'architecture SURF. Le SA comporte les élément suivants : une interface utilisateur (*User Interface*), un module de composition de plan (*Plan Composition Module*), un module de découverte de connaissances (*Knowledge Discovery Module*), un module d'exécution des plans (*Plan Execution Module*) et une pile de communication pour les services Web, incluant SOAP, XML et HTTP.

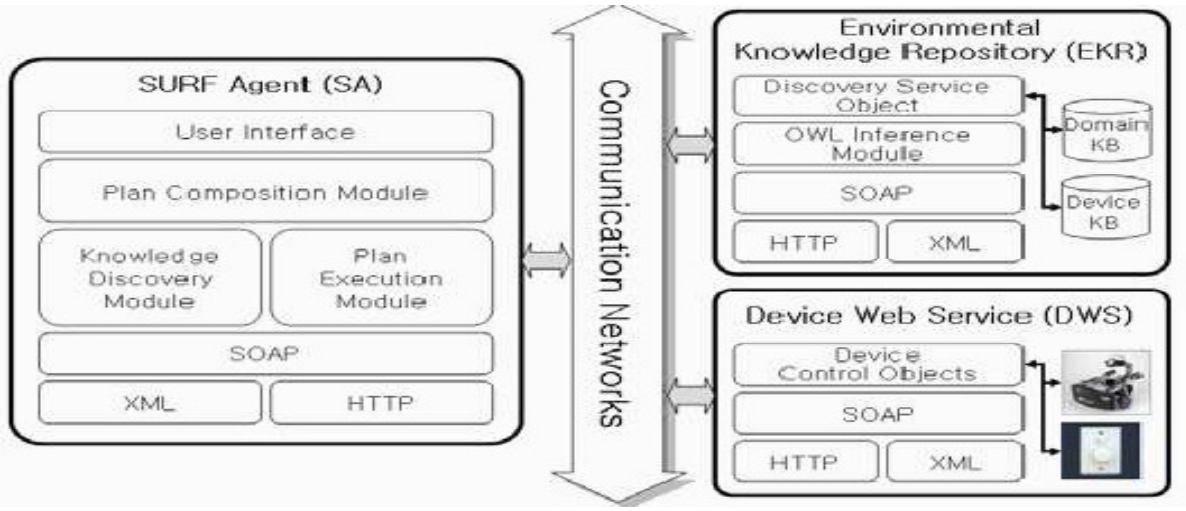


Figure 4.6 : Architecture détaillée de SURF **[163]**.

Le DWS est l'implémentation des services Web pour dispositifs, incluant des robots, des capteurs, etc. Chaque DWS peut avoir des objets de contrôle pour un ou plusieurs dispositifs qui peuvent travailler en collaboration. DWS a également une pile de communication des services Web pour communiquer avec le SA. EKR contient les bases de connaissances (KBs) sur le domaine des services (*Service Domain KBs*) et sur les dispositifs (*Device KBs*) qui sont utilisés dans les phases découverte automatique, composition et exécution. EKR comprend l'objet de découverte de service (*Discovery Service Object*) pour traiter les requêtes de découverte de connaissances avec des prédicats sémantiques sur des connaissances OWL-S. L'EKR comprend également une pile de communication des services Web, car lui-même fonctionne comme un service Web. Un utilisateur peut entrer une requête de service et éventuellement le contexte initial des services via l'interface utilisateur. Ensuite, le module de composition de plan commence à découvrir les connaissances requises pour la planification par le biais du module de découverte de connaissances et compose automatiquement un plan de service en se basant sur la requête de service en entrée et le contexte initial. Le module de découverte de connaissances crée des requêtes sémantiques pour la recherche, dans les bases

de connaissances (KBs) d'EKR, des connaissances sur la planification décrite en OWL-S. Le module d'exécution des plans exécute le plan de service généré via la pile de communication des services Web.

4.2.3.2. Modèle des connaissances

SURF emploie les technologies des services Web sémantique **[266]** et les techniques de planification à base d'IA afin de supporter l'interopérabilité entre les robots et les dispositifs dans les environnements ubiquitaires. Les services Web **[267]** sont implémentés comme méthode d'interface unifiée afin d'accéder aux robots, aux capteurs ainsi qu'à d'autres dispositifs. Les connaissances sur les services Web sont décrites en OWL-S (*Web Ontology Language for Services*) **[85]** qui est un langage de description sémantique des services Web. En général, les descriptions des services pour dispositifs peuvent être fournies par les fabricants de dispositifs ou par les prestataires de services. Les services pour dispositifs sont décrits comme des processus OWL-S atomiques **[85]** et utilisés pour générer l'espace d'actions faisables pour la composition des plans dans les environnements des services. La connaissance sur le modèle commun des services peut être créée par des experts du domaine spécifique de service. Le modèle commun des services est décrit comme des processus OWL-S composites **[85]** et utilisé pour spécifier la façon avec laquelle les plans de service sont composés indépendamment des environnements des services. La base de connaissance du domaine de service (*Service Domain KB*) sauvegarde l'ontologie OWL-S des concepts généraux des services (entrées, sorties, pré-conditions et effets) et les processus composites avec des flux de données internes décrivant les modèles communs des services pour un domaine de service. La base de connaissance des dispositifs (*Device KB*) sauvegarde l'ontologie OWL-S des processus atomiques et les services *grounding* qui correspondent aux dispositifs qui appartiennent aux environnements des services. En d'autres termes, *Device KB* contient des connaissances OWL-S sur les services Web. Ces connaissances concernent le modèle de processus (*process model*) et le modèle de service *grounding* qui relie le modèle de processus avec l'implémentation du service Web. L'interface utilisateur est fournie avec SURF pour permettre à l'utilisateur d'entrée sa requête de service et éventuellement le contexte initial. La localisation de l'utilisateur est considérée comme un contexte essentiel.

4.2.3.3. Modèle de *matching* de services

L'approche de *matching* est utilisée lors de la phase de découverte des connaissances afin d'étendre l'ensemble des services pour dispositifs nécessaires pour la planification. Cette approche est basée sur un raisonnement sur la sémantique du modèle de connaissances des services incluant l'IOPE (entrées, sorties, pré-conditions et effets). Il définit trois relations sémantiques dans la syntaxe de la logique des prédicats. La sémantique de chaque prédicat logique et ses arguments est basée sur la sémantique formelle de RDF **[268]**, RDF Schema **[80]**, OWL **[269]** et de l'ontologie des processus OWL-S **[85]**.

4.2.3.4. Modèle de composition de services

SURF compose automatiquement les plans de service pour répondre à la requête de l'utilisateur en utilisant la planification HTN (*Hierarchical Task Network)* **[270]**. Plus précisément, il se base sur le système SHOP2 **[271]** qui est un planificateur HTN indépendant du domaine. Les connaissances sur le modèle de service sont décrites comme un processus

OWL-S composite et traduites en un ensemble de méthodes de planification SHOP2 selon les structures de contrôle du processus composite décrivant ces connaissances. L'algorithme de traduction (translation) est implémenté dans le module de composition de plan (*Plan Composition Module*) de l'agent SA. Avant d'entamer cette procédure de traduction pour la composition d'un plan faisable, l'ensemble des services pour dispositifs nécessaires à la planification est étendu lors de la phase de découverte de connaissances. L'algorithme d'extension de ces connaissances est basé sur les trois relations sémantiques prédéfinies dans l'approche de *matching*.

4.2.3.5. Modèle de monitoring de services

Le résultat de la phase composition de plan est une séquence de tâches primitives ; c'est à dire une séquence de processus OWL-S atomiques incluant les services *groundings*. Pour être exécuté, un plan de service doit être traduit dans un format exécutable comme un processus de services Web concret. Dans SURF, BPEL4WS (*Business Process Execution Language for Web Services*) **[272]** est utilisé pour créer des processus exécutables de service Web lors de la phase d'exécution du processus. Les flux de données et les connaissances sur le service *grounding* sont utilisés pour générer un processus de service exécutable à partir du résultat de la composition. En outre, SURF est réactif aux changements de la localisation de l'utilisateur.

4.2.3.6. Modèle d'évaluation

L'expérience sur un système SURF prototypique est réalisée dans une maison qui contient une chambre, une cuisine et un salon. Cet environnement expérimental contient les dispositifs suivants : un robot Scorpion (*PC-based ERSP Scorpion robot*) **[273]** avec une connexion USB et IR (infrarouge) pour la commande à distance, des dispositifs commandés par IR, un automate contrôleur PLC (*Power Line Communication*) et des dispositifs commandés par PLC. En outre, il ya 3 lecteurs RFID dans chaque endroit afin de capter la localisation de l'utilisateur qui porte une étiquette RFID. L'implémentation de ce prototype SURF est constituée d'un hôte DWS, d'un serveur EKR, d'un PC comme agent SURF et d'un ordinateur portable monté sur le robot Scorpion. Tous ces éléments sont reliés entre eux via un réseau local filaire ou sans fil. Java2 SDK, HP's JENA sont utilisés comme analyseur OWL et Apache AXIS Web Services toolkit pour implémenter SA, DWS et EKR. Dans le premier scénario de l'expérience, l'utilisateur, porteur d'une étiquette RFID, est assis sur le canapé dans le salon et le robot Scorpion se trouve dans la cuisine. L'utilisateur commande le robot par la requête "rendre plus lumineux" via l'interface utilisateur de l'agent SURF. Comme résultat, le robot se déplace vers le salon et soulève le rideau de la fenêtre. Le deuxième scénario est le même que le premier, mais l'utilisateur a changé de position et se trouve désormais à la cuisine. Comme résultat, le robot se déplace à la cuisine et allume la lumière à l'aide de son contrôleur IR à distance au lieu de soulever le rideau de la fenêtre.

4.2.3.7. Modèle de communication

Le module d'exécution de plan exécute le plan de service généré via la pile de communication des services Web incluant le protocole SOAP, XML et HTTP. L'agent SURF, illustré sur la **figure 4.7**, interagit automatiquement avec des robots, des capteurs et d'autres dispositifs via SOAP selon le plan de service. Ainsi, les robots peuvent être intégrés

automatiquement dans les environnements de services en se basant sur la connaissance des dispositifs de ces environnements.

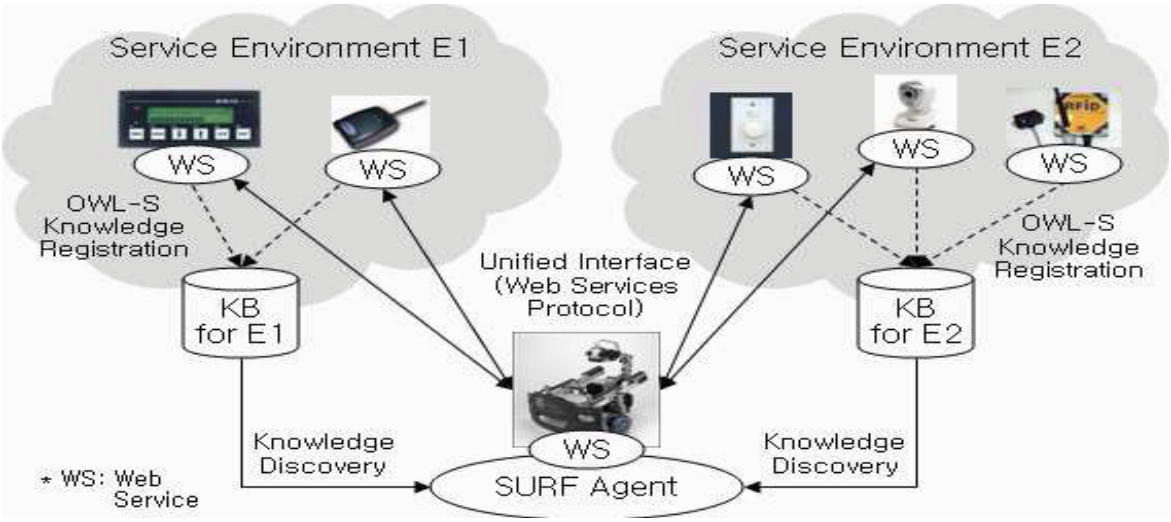


Figure 4.7: SURF Approche de communication **[163]**.

4.2.4. MySIM

4.2.4.1. Architecture générale

Dans **[169]**, les auteurs proposent MySIM (*spontaneous service integration middleware*), un Framework adapté aux environnements *pervasifs* pour l'intégration spontanée des services. Comme le montrent la **figure 4.8**, MySIM est composé de plusieurs services, regroupés sous quatre modules: le module de translation (*Translator module*), responsable de la transformation des services ; le module de génération (*Generator module*), responsable des relations fonctionnels entre les services (composition) ; le module d'évaluation (*Evaluator module*), chargé d'évaluer la qualité de service des propriétés non fonctionnelles des compositions de service ; et le module d'exécution (*Builder module*), chargé de l'exécution des services composites (intégration de services), de l'enregistrement et du monitoring des services.

Le service de translation (*Translator Service*) transforme les diverses technologies des services disponibles dans l'environnement en un modèle de service générique défini. Les règles de transformation extraient et identifient les trois parties principales d'un service: (1) l'interface par l'extraction des signatures des opérations et les descriptions sémantiques, (2) les implémentations par l'extraction des implémentations des opérations, et (3) la qualité de service des propriétés non fonctionnelles par l'extraction des propriétés non fonctionnelles des opérations. Le service de génération (*Generator Service*) est responsable du *matching* syntaxique (signatures d'opération) et sémantique des interfaces fonctionnelles (ensemble d'opérations) des services en vue de les composer ou les adapter. Le service de la *QoS* (*QoS Service)* est responsable d'évaluer les compositions de service obtenues en analysant les propriétés non-fonctionnelles (*QoS*) des services. Le service de décision (*Decision Service*) est basé sur les événements d'apparition et de disparition des services. Il utilise le service de génération et le service de la *QoS* afin de construire toutes les relations d'équivalences abstraites (les compositions compatibles) possibles entre les services. Le service d'exécution

(*Builder Service*) implémente les compositions fournies par le service de génération et le service de la *QoS* et approuvées par le service de décision. Ainsi, l'intégration des services est techniquement réalisée par le service d'exécution. Le service d'enregistrement (*Registry Service*) enregistre les interfaces des services nouvellement transformées et/ou composées dans l'environnement. Ce service effectue également le monitoring des services composites car ils dépendent fortement des services qu'ils intègrent.

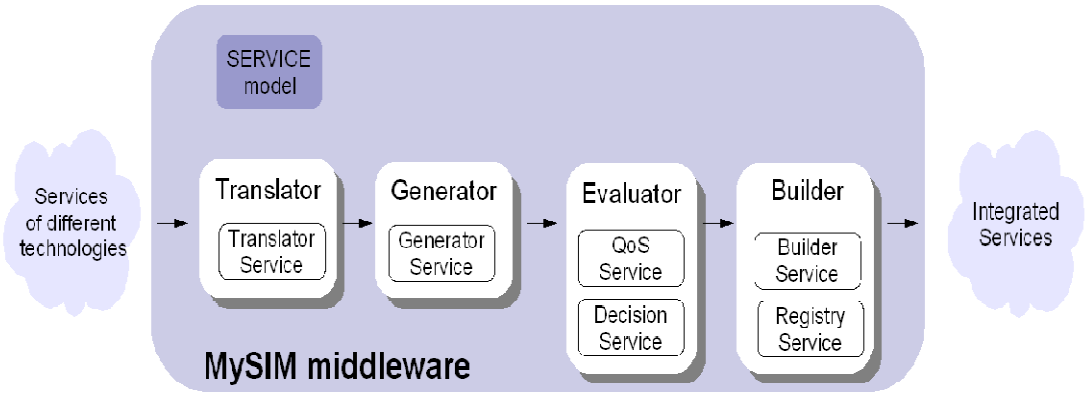


Figure 4.8 : Le middleware MySIM **[169]**.

4.2.4.2. Modèle des connaissances

Dans MySIM, un service peut effectuer un ensemble d'opérations. Une opération est décrite par un concept, un ensemble d'entrées, une sortie et un ensemble de propriétés non fonctionnelles. Chaque service est décrit par un modèle générique de service qui est composé de trois parties principales: une interface, une implémentation et des propriétés non fonctionnelles de la *QoS*. Une interface fonctionnelle précise les opérations qui peuvent être effectuées sur le service. L'implémentation est la mise en œuvre des opérations définies dans l'interface fonctionnelle. Les propriétés non-fonctionnelles de la *QoS* reflètent la qualité de fonctionnement attendue du service qui peut être qualitative ou quantitative. Les opérations et les propriétés de la *QoS* sont également sémantiquement décrites en utilisant les concepts d'une ontologie commune. Les services décrits dans d'autres technologies de services devraient être traduits dans le modèle générique de service défini avant leur utilisation. Cette traduction est basée sur des règles de transformation qui identifient les trois parties principales d'un service: l'interface, les implémentations et les propriétés non-fonctionnelles.

4.2.4.3. Modèle d'évaluation de la *QoS* et du contexte

Une métrique, appelée le degré d'équivalence non-fonctionnelle, est définie pour mesurer le degré de similitudes non-fonctionnelles (*QoS*) entre deux opérations fonctionnellement équivalentes. Le degré d'équivalence non-fonctionnelle, noté $QoS_{Degree}(op_i; op_j)$, entre deux opérations op_i et op_j fonctionnellement équivalentes est évalué sur la base des similitudes de leurs propriétés quantitatives et qualitatives. La normalisation Z-score est utilisée afin de fournir un moyen de comparaison des différentes propriétés non-fonctionnelles en tenant compte de leurs distributions respectives. La distance sémantique pour comparer les propriétés qualitatives retourne une valeur normalisée comprise entre 0 et 1. Plus les concepts de qualité de service sont proches (selon la profondeur des associations parents/fils dans l'ontologie), plus la valeur de la distance sémantique est proche de zéro.

4.2.4.4. Modèle de *matching* de services

Deux services sont syntaxiquement (ou sémantiquement) équivalents s'ils ont des interfaces (i.e. une série d'opérations) syntaxiquement (ou sémantiquement) équivalentes. Deux opérations sont syntaxiquement (ou sémantiquement) équivalentes si respectivement leurs entrées, sorties et leur concept sont syntaxiquement (ou sémantiquement) équivalents. L'équivalence syntaxique est une simple équivalence de type pour les entrées et les sorties et une simple équivalence de nom pour le concept. L'équivalence sémantique repose sur la méthode du *matching* sémantique proposée dans **[274]**. Cette méthode définit quatre niveaux pour le *matching* sémantique entre deux concepts (exacte, *plug in*, subsume et échec). Les auteurs stipulent qu'un concept peut substituer un autre si des relations de *matching* exacte ou *plug in* sont définies entre les deux concepts. Deux services sont composables si syntaxiquement (ou sémantiquement) la sortie d'une opération d'un service peut être utilisée comme une entrée pour une opération de l'autre service. Pour l'équivalence syntaxique, le type de la sortie doit être du même type ou un sous-type du type de l'entrée. Tandis que pour l'équivalence sémantique, le concept de la sortie doit être exacte ou *plug in* dans la relation de *matching* avec le concept de l'entrée.

4.2.4.5. Modèle de sélection de services

La sélection des services est effectuée par le service de la QoS (*QoS Service*) pour vérifier une composition de service et effectuer une adaptation du service composite. Dans l'étape de composition de service, le service de génération (*Generator Service*) retourne toutes les relations fonctionnelles possibles entre les services sans tenir compte de leurs propriétés non-fonctionnels. Toutefois, cela peut conduire à une composition de service non-exécutable si les propriétés non-fonctionnelles de qualité de service des entrées et sorties des opérations impliquées dans la composition ne sont pas compatibles. Pour cela, le service de la *QoS* a besoin de vérifier pour chaque composition de service la compatibilité des opérations en analysant leurs propriétés non-fonctionnelles de qualité de service. Dans l'étape d'adaptation du service composite, un service est remplacé par un autre, offrant des fonctionnalités équivalentes mais avec des meilleures propriétés de qualité de service. Le *QoS Service* assure une bonne substitution entre des services équivalents en choisissant les meilleurs services pour une application donnée en se basant sur la métrique définie dans la **section 4.2.4.3** de ce présent chapitre. Cette métrique permet de mesurer le degré d'équivalence non-fonctionnelle entre les services.

4.2.4.6. Modèle de composition de services

Dans le processus de composition, le *Generator Service* trouve, pour chaque service, tous les services qui répondent à une relation de composition syntaxique ou sémantique (en utilisant le *matching* sémantique) entre les parties fonctionnelles des services. De toutes ces combinaisons, le *Generator Service* ne garde que les services qui possèdent une interface équivalente à un service déjà existant dans l'environnement. Cette combinaison des relations de composition et d'équivalence d'interfaces garantit une composition transparente pour les utilisateurs et les applications qui utilisent les services de l'environnement. En effet, les nouveaux services composites continuent à publier des interfaces bien connues avec des nouvelles implémentations correspondant à la composition de service.

4.2.4.7. Modèle de monitoring de services

Le service d'enregistrement (*Registry Service*) enregistre les interfaces des services nouvellement transformées et/ou composées dans l'environnement. Ce service effectue également le monitoring des services composites qui dépendent fortement des services qu'ils intègrent. Le service d'enregistrement vérifie périodiquement si ces services s'exécutent correctement. Lorsque c'est nécessaire, il peut suspendre, arrêter et démarrer les services. L'accès à un service composite est parfois impossible lorsque l'un de ses services constituant s'avère indisponible. Dans ce cas, le nouveau service composite ne s'exécute pas correctement et le *Registry Service* détecte ce changement. Le *Registry Service* notifie le *Decision Service* des événements liés à l'apparition de service (inscription d'un nouveau service) et la disparition de service (désinscription ou arrêt d'un service). Une adaptation spontanée du service composite peut se produire lors de l'apparition et/ou de la disparition des services. Dans le cas d'une disparition, le *Decision Service* demande au *Generator Service* et le *QoS Service* de lui fournir tous les services disponibles qui sont équivalents (syntaxiquement ou sémantiquement) au service disparu. Le *Decision Service* choisit parmi tous les services disponibles et équivalents, atomiques ou composés, le plus approprié en termes de propriétés de la *QoS* qu'il offre. Il utilise la fonction QoS_{Degree} définie dans **la section 4.2.4.3** pour calculer le meilleur service pour remplacer le service qui vient de disparaître. Dans le cas d'apparition d'un nouveau service, *Decision Service* vérifie via le *Generator Service* si d'autres services, équivalents au nouveau service apparu, sont utilisés par les applications. Si c'est le cas, en se basant sur la fonction QoS_{Degree} proposée par le *QoS Service*, le *Generator Service* évalue, pour chaque application concernée, si le nouveau service répond mieux au besoin de cette application en termes de qualité de service que ses propres services déjà utilisés. Si c'est le cas, une substitution transparence des services est alors effectuée.

4.2.4.8. Modèle d'évaluation

Dans **[169]**, les auteurs présentent un cas d'utilisation simple, appelé *MyStudio,* pour permettre de comprendre et d'illustrer les éléments clés de l'approche proposée pour l'intégration spontanée des services. *MyStudio* est composé des cinq services suivants: webcam, stockage, imprimante, impression et redimensionnement. Ces services offrent des fonctionnalités diverses avec différentes propriétés non-fonctionnelles pour les utilisateurs et les applications. Compte tenu de ces services, une composition spontanée est possible en combinant deux services ensemble, par exemple, le redimensionnement et le stockage, même si cette composition n'est pas requise par les utilisateurs. Le nouveau service composite contient alors le service de stockage étendu par le service de redimensionnement. Les utilisateurs auront ainsi accès à deux services de stockage via des interfaces similaires (interface déjà disponible du service de stockage) mais avec des implémentations différentes. Le premier service permet seulement de stocker une image dans un emplacement défini alors que le deuxième ne permet pas seulement de stocker l'image, mais aussi de la redimensionner selon la commodité de l'utilisateur. La composition spontanée déploie alors un nouveau service dans l'environnement pour l'utilisateur. Une adaptation spontanée est également possible. Si l'un des services d'impression n'est pas disponible, le Framework propose alors le service de l'autre imprimante, spontanément et de manière transparente pour les utilisateurs, car les deux services publient des interfaces similaires.

4.2.5. MEDUSA

4.2.5.1. Architecture générale

Dans **[174]**, les auteurs présentent MEDUSA, un Framework permettant la composition d'applications orientées utilisateur dans les espaces intelligents. Les principaux objectifs de MEDUSA consistent à fournir aux utilisateurs finaux un support d'abord pour la création et la personnalisation de leurs applications et ensuite pour contrôler le processus de composition en fonction de leurs besoins. Pour atteindre ces objectifs, MEDUSA utilise des outils de composition et un ensemble d'interfaces de contrôle pour la création d'applications des utilisateurs. MEDUSA gère aussi l'interopérabilité entre les dispositifs, les réseaux et les plates-formes hétérogènes. L'architecture de MEDUSA est décomposée en trois couches, comme le montre la **figure 4.9**: la couche d'interopérabilité de la communication (*Communication interoperability layer*), la couche d'interopérabilité des services (*Service interoperability layer*) et la couche centrée sur l'utilisateur (*User-centric layer*). La couche d'interopérabilité de la communication utilise une interface réseau commune pour cacher les technologies sous-jacentes des réseaux multi-radio. Cette couche gère les réseaux multi-radio (*multi-radio networking*) et les réseaux multi-protocole (*multi-protocol network*) **[275]**. La couche d'interopérabilité des services permet l'interopérabilité sémantique et syntaxique entre les différents clients et fournisseurs des services sans imposer l'utilisation d'une norme spécifique. L'interopérabilité est réalisée en utilisant un modèle commun de description des services, qui spécifie la fonction de correspondance (translation) entre les concepts des services et les fonctionnalités fournies par les nœuds de la plate-forme **[93]**. En outre, cette couche est responsable de la découverte des services et de la composition d'applications. La couche centrée sur l'utilisateur fournit des fonctionnalités pour créer et personnaliser des applications et des interfaces. Les utilisateurs finaux peuvent alors utiliser ces fonctionnalités pour contrôler le processus de composition et d'adapter des applications au moment de leurs exécution. L'utilisateur effectue cette adaptation en fournissant les informations sur ses préférences et en choisissant les instances des services qu'il souhaite avoir dans son application.

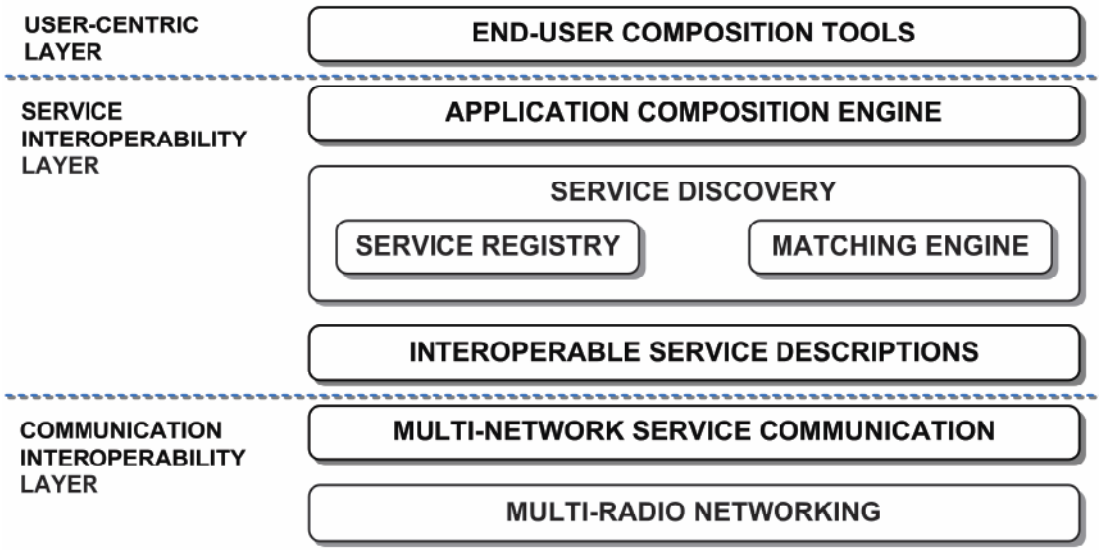


Figure 4.9 : L'architecture conceptuelle de MEDUSA **[174]**.

4.2.5.2. Modèle des connaissances

La couche d'interopérabilité des services est réalisée par l'adoption du modèle de description de service *AmIi* (*Ambient Intelligence interoperability*) **[276]** et la découverte interopérable et ambiante des services **[93]**. *AmIi* est une solution basée sur la sémantique qui réalise une interopérabilité conceptuelle entre les plates-formes hétérogènes des services. Ce type d'interopérabilité est requis par les fournisseurs de services désirant publier leurs services (avec des langages de description différents) en utilisant différents protocoles de découverte de services. *AmIi* s'appuie sur le modèle interopérable de description de service et le moteur de découverte de service multi-protocole. Le modèle interopérable de description de service permet la correspondance entre les langages hétérogènes de description de service, offrant ainsi une conformité au niveau syntaxique et sémantique. En conséquence, les descriptions des services, qui ont été initialement écrites dans des langages tels qu'UPnP ou WSDL, peuvent être traduites dans des descriptions du modèle interopérable correspondant. *AmIi* capture à la fois les propriétés fonctionnelles (par exemple, les interfaces) et non fonctionnelles des services (par exemple, la *QoS*). De plus, *AmIi* fournit des informations sur le comportement des services et sur le service *grounding*. Ce dernier spécifie comment le service est accédé en utilisant un protocole de communication. Le comportement du service est modélisé en utilisant un langage de *workflow* tel que BPEL. Les propriétés fonctionnelles et non-fonctionnelles sont spécifiées soit par une référence à une ontologie existante ou bien par des descriptions sémantiques intégrées à ces propriétés. En outre, les auteurs dans **[174]** supposent que chaque environnement ubiquitaire fournit un ensemble de cartes qui sont associées avec les instances de services disponibles dans l'environnement. Ces cartes peuvent être conçues, par exemple, par l'administrateur des services concrets. Ainsi, chaque service est représenté par une carte en papier carré avec une icône graphique sur une face et une étiquette RFID fixée sur l'autre face. Chaque étiquette RFID contient un lien hypertexte vers sa description de service qui peut être lu en touchant la carte avec un téléphone mobile équipé d'un RFID. Par ailleurs, MEDUSA fournit des interfaces de contrôle différentes qui permettent différents degrés d'intégration des utilisateurs dans le contrôle de la composition d'application.

4.2.5.3. Modèle d'évaluation de la QoS et du contexte

Le moteur de composition d'applications dans MEDUSA utilise les algorithmes de composition initialement introduit dans **[277]**. Ces algorithmes d'optimisation sont basés sur les théories de l'informatique évolutive et génétique, et ils sont utilisés pour optimiser les configurations d'une application (ensemble de services qui constituent l'application). Les algorithmes effectuent l'optimisation sur la base des exigences de l'utilisateur en termes de qualité de service et les critères spécifiés pour l'optimisation. Par exemple, une configuration de l'application peut être optimisée afin de minimiser la consommation globale de l'application en terme de bande passante et de maximiser les propriétés de la *QoS* spécifiée par l'utilisateur. Ces algorithmes sont génériques et supportent (i) les critères personnalisables qui peuvent inclure plusieurs objectifs d'optimisation simultanés et (ii) les différents types de propriétés de la *QoS* d'une application qui peuvent être ajoutées ou supprimées des descriptions des services au moment de l'exécution selon les besoins.

4.2.5.4. Modèle de sélection de services

Lorsque les utilisateurs développent des applications utilisant les outils de composition offerts, ces applications n'existent que sous forme de descriptions abstraites qui doivent être réalisées par les instances de services concrets. En d'autres termes, les services constituant une application doivent être connectés au moment de l'exécution à des instances de service disponibles dans l'environnement. Cette tâche est effectuée par le compositeur d'application qui peut être contrôlé par les utilisateurs utilisant le mécanisme de sélection de services. Dans MEDUSA, la sélection de services est effectuée par les utilisateurs grâce à un ensemble d'interfaces qui offrent deux fonctions: (i) permettre aux utilisateurs de choisir parmi les configurations d'application possibles proposées par le moteur de composition d'application, et (ii) permettre aux utilisateurs de choisir directement les instances des services qui sont réellement disponibles dans l'environnement.

4.2.5.5. Modèle de découverte de services

La couche d'interopérabilité des services est responsable de la découverte des services. Cette couche stocke d'abord toutes les descriptions des services disponibles dans un référentiel. Elle effectue par la suite une recherche dans ce référentiel pour trouver des descriptions de service qui *matchent* une requête de service. Le service de découverte de MEDUSA utilise la découverte de service ambiant et interopérable **[93]** et la fonction de découverte fournie par le modèle *AmIi*. La fonction de découverte de service interopérable fournie par *AmIi* est réalisée en utilisant des référentiels de services distribués qui supportent les protocoles existants de découverte des services. Cela signifie que chaque référentiel gère un ensemble de *plugins* (i.e., *proxys*) associé à divers protocoles de découverte des services. Les protocoles existants de découverte des services sont alors en mesure d'échanger des descriptions de service avec les référentiels et répondre aux demandes de la requête. En outre, la découverte de service ambiant et interopérable prend en charge un *plugin* pour l'enregistrement et l'interrogation des descriptions de service directement dans le format interopérable. Cependant, un tel mécanisme exige la traduction de toutes les descriptions de service dans un format interopérable, ce qui peut augmenter potentiellement la latence du protocole de découverte.

4.2.5.6. Modèle de composition de services

La composition d'application ubiquitaires est conçue par l'utilisateur en utilisant une interface physique à base des RFID. MEDUSA propose dans sa couche centrée sur l'utilisateur un ensemble d'outil pour la composition d'une application par l'utilisateur avant son envoi au moteur de composition d'applications. Au niveau de cette couche, la structure de la composition est fournie au système par l'utilisateur en s'appuyant sur des cartes des services via une interface sur son téléphone mobile. Dans cette composition, l'utilisateur final doit spécifier les dépendances de contrôle et de données entre les services qu'il a choisis. Cette étape peut être supportée par un agent assistant déployé sur le téléphone mobile. Par exemple, si l'utilisateur conçoit une application qui n'a pas une structure complète, l'assistant lui suggère la façon de compléter cette structure en utilisant, par exemple, des informations à partir d'une base de données avec des descriptions d'applications existantes. Une approche similaire est utilisée dans **[278]** pour la programmation des maisons intelligentes. Les

compositions conçues par les utilisateurs sont sous des formes abstraites qui doivent être réalisées par les instances des services concrets. Cette tâche est effectuée par le compositeur d'application qui peut être contrôlé par les utilisateurs via un ensemble d'interfaces. MEDUSA propose quatre types d'interfaces de contrôle, qui permettent différents degrés de participation des utilisateurs dans le contrôle de la composition d'application: interface manuelle, interface semi-manuelle, interface mixte et interface autonome. L'interface manuelle suppose que les utilisateurs ont un contrôle total sur la composition d'application. Ainsi, le moteur de composition d'application n'est pas du tout utilisé. Les utilisateurs peuvent choisir les instances de service en les touchant avec leur téléphone mobile. Dans l'interface semi-manuelle les utilisateurs peuvent sélectionner certains services tandis que le moteur de composition d'application complète la structure de l'application en attribuant automatiquement les services manquants. L'interface mixte restreint le contrôle de l'utilisateur aux options proposées par le compositeur d'application. Toutefois, les utilisateurs peuvent toujours passer à une autre interface de contrôle s'ils ne sont pas satisfaits des options proposées par le système. L'interface autonome utilise le compositeur d'application pour démarrer et gérer une configuration d'application sans l'utilisateur. Cette interface suppose que l'utilisateur ne veut pas contrôler la composition d'application. Une fois spécifiée, la composition est alors exécutée par le moteur de composition d'application qui implémente les algorithmes de composition cités dans **[277]**.

4.2.5.7. Modèle d'évaluation

L'approche proposée a été testée avec un outil prototype de composition et une série initiale de cartes des services. Deux utilisateurs ont participé à l'expérience préliminaire qui a duré une heure et demi. Les utilisateurs ont été invités à concevoir des applications abstraites en utilisant les cartes des services qui sont à leur disposition. En outre, ils ont été autorisés à utiliser d'autres cartes des services qu'ils pensent nécessaires. Au total, six applications ont été conçues pour de multiples environnements, tels que la maison, le bureau et l'hôpital. Plusieurs services supplémentaires ont été également proposés lors de cette expérience.

4.2.5.8. Modèle de communication

La couche d'interopérabilité de la communication (*Communication interoperability layer*) de MEDUSA est réalisée à l'aide du middleware de communication ubiSOAP **[275]**. ubiSOAP est spécialement conçu pour des dispositifs ubiquitaires et portables avec des ressources limitées qui peuvent être interconnectés via de multiples connexions sans fil, tels que Bluetooth, Wi-Fi et GPRS. Cette fonctionnalité est essentielle pour MEDUSA qui est destiné à être utilisé sur des téléphones mobiles. En outre, ubiSOAP adopte le standard des services Web pour implémenter des services ubiquitaires, et étend le protocole standard SOAP en tenant compte de la connectivité et la mobilité des nœuds. La **figure 4.10** montre l'architecture à deux couches d'ubiSOAP. La couche inférieure est la couche de connectivité qui sélectionne le réseau selon les politiques des utilisateurs, car certains utilisateurs peuvent exiger l'utilisation d'un certain type de réseau pour des raisons personnelles. Cette couche identifie les applications dans le réseau de l'environnement. La couche supérieure est la couche de communication qui étend l'utilisation du protocole standard SOAP par la messagerie entre les services participants en introduisant des protocoles de transport SOAP multi-réseaux, multi-radios, point-à-point et de groupe (multicast). ubiSOAP a été implémenté

en utilisant deux moteurs SOAP : Axis27 et CSOAP8. Ceci démontre qu'ubiSOAP permet à des applications existantes de communiquer via le protocole standard SOAP. Cette fonctionnalité garantit la compatibilité de MEDUSA avec plusieurs services existants.

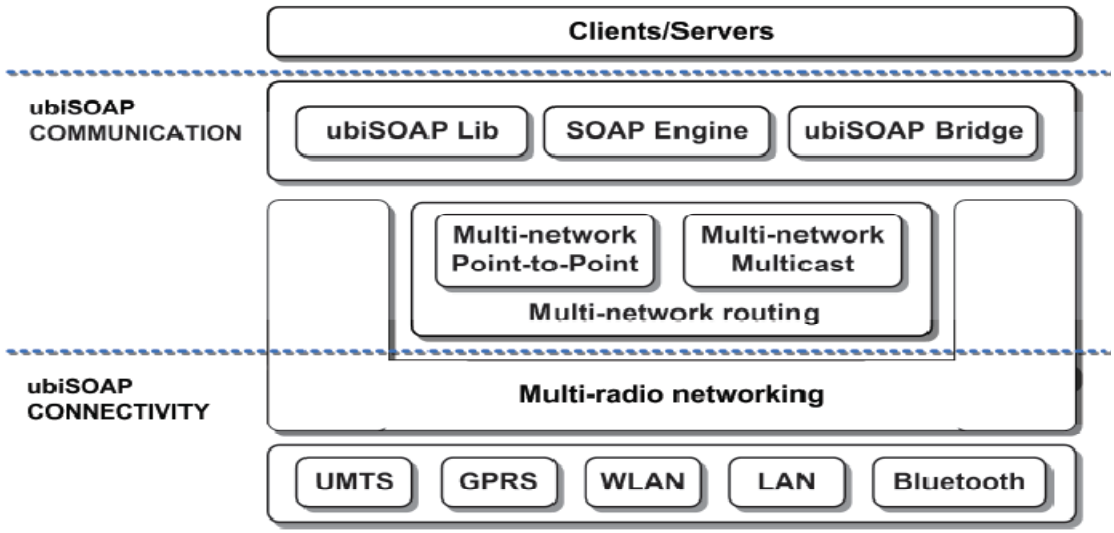


Figure 4.10 : L'architecture ubiSOAP proposée dans **[275]**.

4.2.6. MUSIC

4.2.6.1. Architecture générale

Dans **[171]**, les auteurs présentent MUSIC (*Middleware Support for Self-Adaptation in Ubiquitous and Service-Oriented Environments*), comme un support Middleware pour l'auto-adaptation dans des environnements ubiquitaires et orientés service. La version initiale de la plate-forme MUSIC, détaillée dans **[279]**, supporte uniquement l'adaptation des architectures à base de composantes. Le travail effectué dans **[171]** propose une extension complète de la plateforme initiale de MUSIC afin de permettre l'auto-adaptation des applications mobiles et ubiquitaires avec la présence des Architectures Orientée Service (SOA).

L'architecture initiale de MUSIC: La **figure 4.11** illustre l'architecture basée sur des composants de la plate-forme initiale de MUSIC. La planification est typiquement déclenchée par des changements de contexte détectés par le gestionnaire de contexte (*Context Manager*). Le gestionnaire d'adaptation (*Adaptation Manager)* effectue une planification basée sur l'adaptation. Le contrôleur d'adaptation (*Adaptation Controller*) coordonne le processus d'adaptation. Le raisonneur d'adaptation (*Adaptation Reasoner)* supporte l'exécution des heuristiques de planification. Cette exécution est conduite par des métadonnées incluses dans les plans **[280]**. Le référentiel de plans (*Plan Repository)* fournit l'interface *IPlanResolver* pour le raisonneur d'adaptation permettant ainsi la récupération récursive des plans associés à un port donné. Toutes les métadonnées supplémentaires aideront le référentiel de plans à sélectionner certains plans et donc à réduire considérablement l'espace d'exploration **[280, 281]**. Le raisonneur d'adaptation construit une configuration valide de l'application et écarte celles dont les dépendances ne sont pas résolues. Puis, les heuristiques classent les configurations de l'application par l'évaluation de leurs utilités respectives en se basant sur leurs propriétés dont les valeurs sont récupérées à partir du gestionnaire de la QoS (*QoS Manager*). Le processus de reconfiguration est géré par l'exécuteur de configuration

(Configuration Executor), qui utilise l'ensemble des plans choisis par le planificateur afin de reconfigurer l'application.

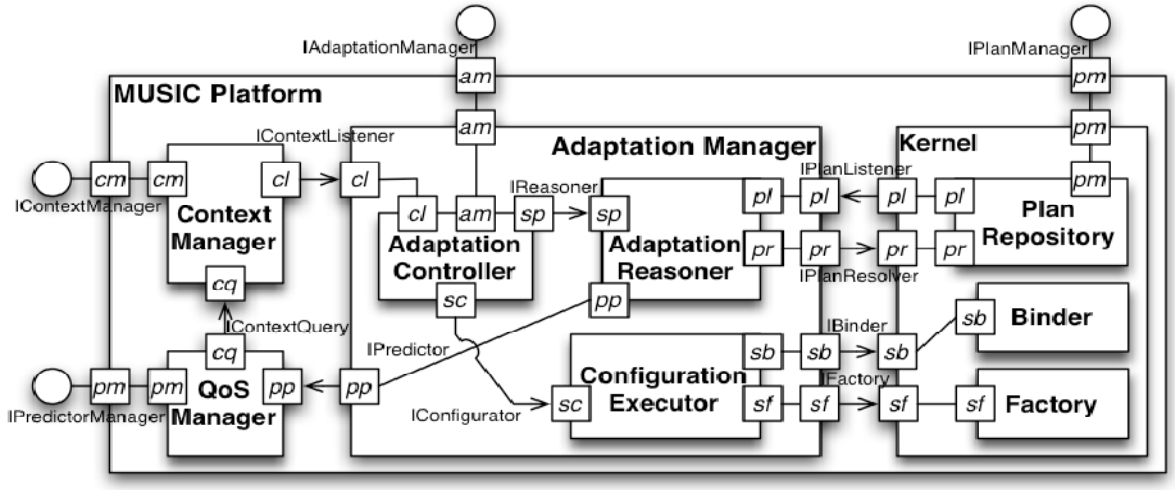


Figure 4.11 : L'architecture de la plateforme MUSIC **[171]**.

Architecture de la plate-forme orientée service de MUSIC: La plateforme initiale de MUSIC présentée précédemment se concentre sur des systèmes d'auto-adaptation basés sur des composants. Cette plateforme a été étendue afin de supporter les principes des architectures SOA ainsi que la réalisation d'une implémentation de référence pour MUSIC en intégrant de nouveaux composants dans la plateforme initiale (**Figure 4.12**). Plus précisément, le service de découverte (*Service Discovery*) est responsable de la publication et de la découverte des services en utilisant différents protocoles de découverte. Il est également responsable de la création des plans des services basés sur les descriptions de ces services et les accords négociés par la négociation SLA (*SLA Negotiation*). Le monitoring SLA (*SLA monitoring*) est responsable de la vérification de l'état de ces accords et de la prise des mesures appropriées en cas de leur violation. Le service d'accès distant (*Remoting Service*) est responsable de l'exportation des services du côté fournisseur de services, et de la liaison à ces services du côté client de services. Chaque fois qu'un service est exporté, il permettra d'accepter les requêtes des clients de services à distance.

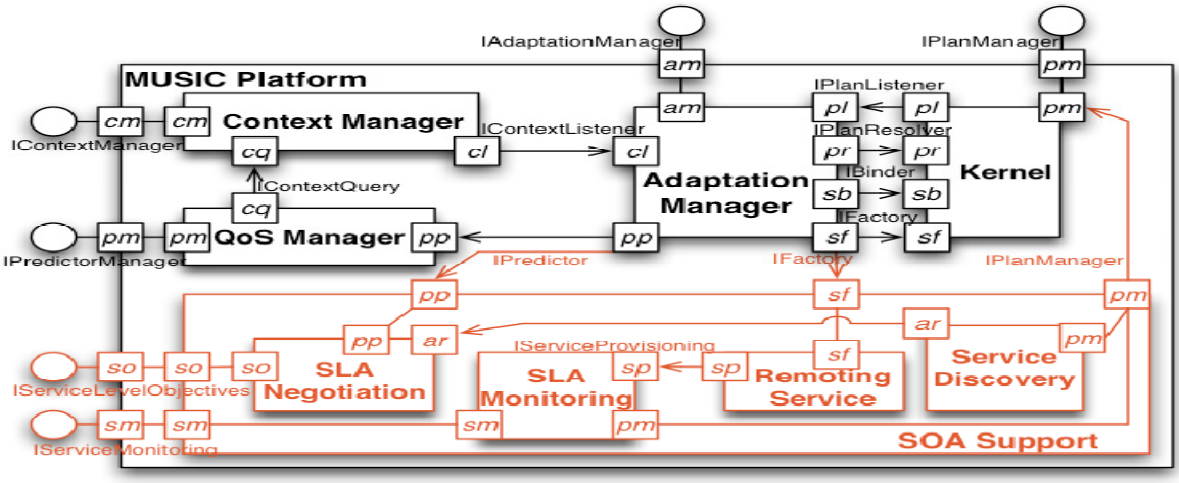


Figure 4.12 : La configuration SOA de la plateforme MUSIC **[171]**.

4.2.6.2. Modèle des connaissances

Les connaissances dans MUSIC sont décrites dans un modèle sensible à la *QoS* (*QoS-aware model*). Ce modèle permet la description des compositions abstraites, des dimensions pertinentes de la *QoS* et du comment ces dimensions sont affectées lors de la variation de la configuration d'une composition de services. Le modèle sensible à la *QoS* décrit la composition abstraite comme un ensemble de rôles qui collaborent à travers des ports, qui représentent soit les fonctionnalités fournies ou requises par les composants collaborateurs. Les propriétés et les fonctions prédictives de propriété (*property predictor*) associées aux ports décrivent les relations entre les différents composants collaborateurs en termes de propriétés de la *QoS*. Un port a un type qui définit la fonctionnalité représentée par le port en termes d'interfaces et de protocoles. Une réalisation d'un composant implémente les ports de ce composant et peut être atomique ou composite. Une réalisation composite est une composition abstraite qui permet une décomposition récursive. Les contraintes sont des prédicats sur les propriétés des composants constituant une composition abstraite. Ces contraintes permettent de restreindre l'espace des combinaisons possibles pour la réalisation de la composition **[282, 283]**. Chaque description d'un service renferme le type de ce service ainsi qu'un modèle d'accord (*agreement template*) **[284]** décrivant les propriétés statiques de la *QoS* fournie par ce service. Les dimensions de qualité de service se référant à des propriétés dynamiques de l'application sont calculées ultérieurement en fonction des capacités et de la charge des nœuds du réseau.

4.2.6.3. Modèle d'évaluation de la QoS et du contexte

A chaque dimension de qualité de service est associée une fonction d'utilité qui mesure la satisfaction de l'utilisateur par rapport à cette dimension. La fonction d'utilité globale d'une application est fournie par le développeur et elle est typiquement exprimée comme une somme pondérée de l'ensemble des fonctions d'utilité relatives à chaque dimension de qualité de service où les poids expriment les préférences des utilisateurs par rapport à ces dimensions. Dans MUSIC, la négociation pour les propriétés de la *QoS* statiques (par exemple, le coût des services) est généralement effectuée au cours de la découverte des services. Les valeurs des propriétés statiques de la *QoS* sont incluses dans le plan de service pour une utilisation ultérieure. La négociation dynamique est particulièrement nécessaire lorsque le gestionnaire d'adaptation (*adaptation manager*) a besoin de raisonner sur les propriétés de la *QoS* mise à jour (par exemple, la précision du service en cours). Dans ce cas, ces propriétés sont évaluées par des heuristiques de raisonnement. Ces propriétés constituent alors les entrées de la fonction d'utilité normalisée qui calcule l'utilité espérée de la configuration évaluée **[279, 280, 282]**.

4.2.6.4. Modèle de sélection de services

Dans MUSIC, la planification se réfère au processus de sélection des composants d'une configuration qui fournissent la meilleure utilité à l'utilisateur final. Ce processus sera déclenché au démarrage de l'application et au moment de l'exécution lorsqu'il y a un changement dans le contexte d'exécution. Quand un tel processus d'adaptation est déclenché, le middleware évalue tous les plans associés aux rôles de la composition abstraite. Pour chaque plan, il résout les dépendances de planification et évalue la pertinence de sa

configuration par rapport au contexte actuel d'exécution en calculant ces propriétés de qualité de service et son utilité.

4.2.6.5. Modèle de découverte de services

Les fournisseurs de services rendent leurs services accessibles par des clients conformément aux protocoles spécifiques de découverte des services. La plate-forme MUSIC supporte un ensemble extensible de protocoles de découverte permettant la détection des services disponibles dans l'environnement. La découverte d'un service consiste à récupérer sa description, qui comprend des informations sur les capacités du service, sa sémantique, et ses propriétés de la QoS sous forme d'un modèle d'accord (*agreement template*). Le service de découverte (*Service Discovery*) est responsable de la publication et la découverte des services utilisant des protocoles différents. Le service de découverte publie chaque description de service définissant les fonctionnalités offertes et contenant les informations nécessaires pour que le client puisse y accéder. Si le fournisseur de services offre des garanties supplémentaires pour les services publiés, les modèles d'accord sont alors publiés en plus de la description du service. Le service de découverte supporte également l'enregistrement dynamique des récepteurs de découverte (*discovery listeners*). Un récepteur de découverte s'enregistre auprès du service de découverte pour être informé de la découverte des services auxquels il a un intérêt particulier. Une fois un service est découvert, le service de découverte informe tous les récepteurs de découverte enregistrés et concernés par ce service en leur transmettant sa description.

4.2.6.6. Modèle de composition de services

Le modèle sensible à la QoS (*QoS-aware model*) qui est adopté décrit une composition abstraite comme un ensemble de rôles qui collaborent à travers des ports. Ces derniers représentent soit les fonctionnalités fournies ou requises par les composants collaborateurs. À l'exécution, ce modèle est représenté comme des plans dans le Framework. Un plan reflète une réalisation d'un composant et décrit ses ports, ses propriétés ainsi que les dépendances implicites avec sa plate-forme d'exécution (par exemple, le type de plateforme et sa version). Dans le cas d'une réalisation d'un composant atomique, le plan contient aussi une référence à la classe qui réalise ce composant. Dans le cas d'une réalisation d'un composant composite, le plan décrit la structure interne en termes de rôles et de ports ainsi que les connexions entre eux. Les variations sur le plan d'un composant composite sont obtenues en décrivant un ensemble de réalisations alternatives possibles des rôles.

4.2.6.7. Modèle de monitoring de services

L'exécuteur de configuration (*Configuration Executor*) parcourt généralement tous les plans qui composent une nouvelle configuration afin de reconfigurer l'application. L'exécuteur de configuration distingue entre les plans qui se rapportent à des services disponibles et les plans qui se rapportent à des services qui ne sont pas encore disponibles. Afin de bénéficier des services distants, l'exécuteur de configuration est confronté à un troisième cas: si le plan se rapporte à un service distant disponible dans l'environnement, l'exécuteur de configuration utilise le service d'accès distant (*Remoting Service*) pour générer un composant spécifique qui agira comme un *proxy* de service. Un *proxy* de service est un représentant local du service distant. En particulier, il implémente le type de service décrit par les composants de

l'application et encapsule la communication nécessaire pour accéder au service à distance. En invoquant le proxy de service, un client de service interagit avec le service à distance de façon transparente comme si le service distant est local. Pour des raisons de monitoring, le *proxy* de service est instrumenté avec des mécanismes de monitoring appropriés par le composant monitoring SLA (*SLA Monitoring*) selon le contenu de la SLA (par exemple, le délai de réponse et la qualité des résultats). *SLA Monitoring* est responsable de la vérification de l'état des SLA et de la prise des mesures appropriées en cas de violation des accords spécifiés.

4.2.6.8. Modèle d'évaluation

Ce travail présente un scénario d'un utilisateur qui est en voyage pour rencontrer un ami. Plusieurs services sont nécessaires afin de planifier ce voyage, à savoir un service d'itinéraire pour le métro, un service de localisation et un service de cartographie. Ce scénario commence lorsque l'utilisateur entre dans le métro Parisien. La société d'exploitation du métro (RATP) offre un service d'itinéraire pour le transport public et un service de cartographie de Paris à deux niveaux de qualité: basique et premium. Il y a aussi un accès via UMTS à des services commerciaux de haute qualité, mais avec un coût plus élevé. Lorsque l'utilisateur sollicite un service de bonne qualité, son dispositif choisit le service premium fourni par la RATP qui est moins cher. Pendant le voyage, il y a eu un incident dans le métro et le blocage de l'itinéraire initialement prévu. L'assistant de voyage informe alors l'utilisateur et lui propose un itinéraire de métro alternatif avec une autre station finale. De plus, la qualité du service de cartographie se dégrade car la RATP réserve une part importante de sa bande passante pour guider le personnel d'urgence. En outre, l'utilisateur ne peut pas utiliser le service de localisation par GPS car les satellites de ce système sont hors de vue à l'intérieur du métro. Comme meilleure solution, le dispositif de l'utilisateur choisit alors les services commerciaux externes de haute qualité, et ce malgré leurs coût élevé.

4.2.6.9. Modèle de communication

MUSIC est implémenté sur la plateforme OSGi. Le choix de cette plateforme est effectué selon certain critères, en particulier, une plateforme open-source, tient compte des ressources limitées des dispositifs, et supporte l'approche orienté service (SOA). Le service de découverte de MUSIC (*Service Discovery*) délègue les requêtes de publication et de découverte des services à des implémentations spécifiques du protocole de découverte des services. Ces implémentations sont connectées (*plugged*) au Framework MUSIC, comme des services OSGi qui implémentent l'interface *IServiceDiscoveryFactory*. MUSIC supporte également plusieurs autres protocoles notamment, le protocole SLP (*Service Location Protocol*) basé sur jSLP **[285]** et le protocole UPnP (*Universal Plug and Play*) basé sur le bundle UPnP de *DomoWare* **[286]**.

4.2.7. SeSCO

4.2.7.1. Architecture générale

Dans **[92]**, les auteurs étendent le Framework SeSCO (*Seamless Service Composition in Pervasive Environments*) qui est initialement présenté dans **[287, 288]**. Ils présentent un nouveau mécanisme de composition de service pour l'informatique ubiquitaire en se basant sur la plate-forme middleware orientée services, appelée PICO (*Pervasive Information*

Communities Organization) **[289]**, afin de modéliser et de représenter les ressources comme des services. Deux approches de composition de service sont présentées dans SeSCO: approche centralisée et approche hiérarchique. Dans l'approche centralisée, tous les services disponibles sont analysés et stockés sous forme d'un graphe d'agrégation sur la base des paramètres sémantiques associés à ces services et leur type syntaxique. Après cette agrégation, la résolution d'une tâche (requête) est effectuée en deux étapes. Dans la première étape, une ou plusieurs compositions possibles sont construites en se basant sur la sémantique des paramètres. Dans la deuxième étape, les services qui peuvent prendre part à la composition sont identifiés sur la base des paramètres syntaxiquement corrects. La composition identifiée est alors stockée dans un graphe de composition, qui est renvoyé comme résultat de la requête soumise. La composition hiérarchique de service, quant à elle, est basée sur le principe de l'organisation des ressources (dispositifs) en utilisant un protocole appelé LATCH.

Cette organisation consiste à classifier les dispositifs de l'environnement ubiquitaire selon leurs ressources. Le principe de base de cette classification consiste à rattacher les dispositifs dotés de ressources limitées à des dispositifs riches en ressources afin d'effectuer leurs opérations. Les dispositifs qui font partie de l'infrastructure et qui ont des contraintes relativement faibles sur leurs ressources sont généralement classés dans le niveau le plus élevé. Des dispositifs, tels que les PCs et les serveurs, peuvent supporter les opérations essentielles, telles que la découverte de service, la composition de services, et le fonctionnement comme *proxy* pour leurs homologues (dispositifs) qui sont limités en ressources. Il en résulte une relation hiérarchique comme le montre la **figure 4.13**. Cette classification est effectuée au moment de la configuration des dispositifs et l'installation du middleware. Quand une tâche particulière (requête) doit être accomplie, en fonction du niveau du dispositif générant cette requête, le processus de résolution de tâches commence soit par le même dispositif générant la requête ou bien par ses parents intermédiaires dans la hiérarchie (arbre) des dispositifs. L'approche proposée prend également en compte l'arrivée et le départ dynamique des dispositifs.

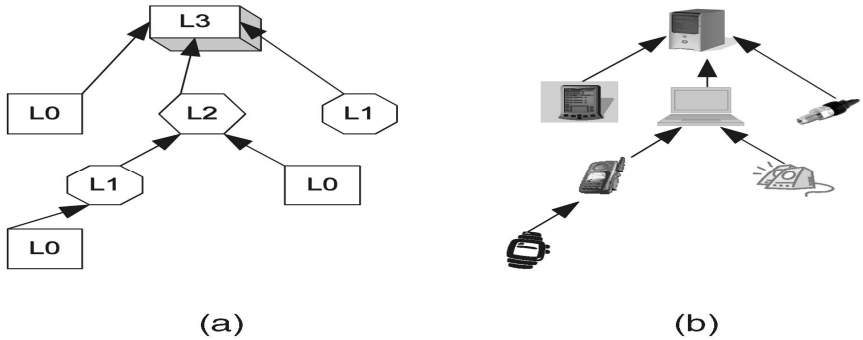


Figure 4.13 : (a) L'organisation hiérarchique des dispositifs. (b) Un exemple de hiérarchie.

4.2.7.2. Modèle des connaissances

Le modèle proposé permet la description des connaissances à la fois sur les services et sur les tâches (requêtes). Un service est décrit comme un graphe simple contenant les entrées, les sorties et les attributs du service. Ces attributs comprennent la description sémantique du service, son emplacement, son coût d'utilisation, son débit promis, son temps d'attente

annoncé, etc. La représentation graphique d'une tâche contient les descriptions des services requis et les attributs représentant les préférences spécifiées, tels que l'emplacement préféré, le coût, le temps d'attente acceptable, etc. Les auteurs supposent que la représentation graphique d'une tâche est initialement intégrée dans l'application par ces concepteurs ou bien l'application possède la capacité de générer le modèle d'une tâche à partir du contexte actuel.

La représentation d'un service: Chaque service (appelé service *delegent*) est décrit comme un graphe simple $Gs = \{Vs, Es, us, \varepsilon s\}$ avec *Vs* étant l'ensemble d'éléments qui représentent le service lui-même. L'ensemble des arcs orientés *Es* représente les entrées et les sorties du service. La fonction *us* est responsable de l'intégration des attributs du service (la description sémantique, l'emplacement, le coût d'utilisation, etc.) sur les sommets. La fonction εs représente les attributs des arcs, tels que la description sémantique d'un paramètre, le type d'un paramètre, le format, etc.

La représentation d'une tâche: chaque tâche est modélisée comme un graphe orienté $Gr = \{Vr, Er, \mu r, \varepsilon r\}$ où l'ensemble des sommets *Vr* représente les services requis pour accomplir la tâche, et l'ensemble des arêtes *Er* représente les transitions entre les services. Les attributs des préférences (coût souhaité, temps d'attente acceptable, etc.) sont représentés sur les sommets par μr, et les attributs sur les arcs sont représentés par εr, qui se charge d'attacher la description sémantique des données, le format des données, les exigences de qualité, etc.

4.2.7.3. Modèle de *matching* de services

Une tâche utilisateur est transmise à l'annuaire pour résolution. Au niveau de l'annuaire, les composants du graphe de la tâche reçue sont *matchés* avec les services basiques disponibles pour générer les services composites possibles. S'il existe un *matching* direct entre un service disponible et les exigences spécifiées dans *Gr*, la tâche utilisateur est alors trivialement résolue. Si un tel *matching* n'existe pas, une combinaison de services disponibles est utilisée afin de répondre aux exigences de la tâche. Cette combinaison est construite de telle manière que le service composite qui en résulte offre les mêmes fonctionnalités que le service requis. Deux services S_1 et S_2 peuvent être combinés pour former un service composite S_3 si et seulement si les conditions suivantes sont vérifiées:

$$Output\ (S_1) \cong Input\ (S_2) \quad (a)$$

$$\varepsilon s(S_2^{in}) > \varepsilon s(S_1^{out}) \quad (b)$$

La première condition (a) veille à ce que la sortie produite par S_1 peut être consommé par S_2. La deuxième condition (b) indique que les attributs de sortie de S_1 peuvent être complètement acceptés par S_2. Alors que (a) restreint les services qui peuvent être combinés en fonction du type de paramètres, (b) impose la condition que les données produites par un service soient consommées par son successeur avec un minimum de pertes.

4.2.7.4. Modèle de composition de services

La composition centralisée de service: dans un système centralisé, tous les services présents au sein d'un environnement ubiquitaire sont inscrits dans un annuaire centralisé. Au niveau de cet annuaire, le graphe *Gs* de chaque service est analysé pour le stockage dans un graphe global d'agrégation *Gp* à deux niveaux. La première étape d'agrégation G_p^1 est basée

sur les paramètres sémantiques associés à chaque service. La deuxième étape d'agrégation G_p^2 est basée sur le type syntaxique des paramètres de chaque service. Un algorithme est proposé pour stocker tous les services enregistrés dans un graphe d'agrégation *Gp*. Durant le processus d'agrégation, les attributs disponibles dans la description des services sont maintenus comme des attributs sur les arcs de G_p^2. Ces attributs sont essentiels pour *matcher* les exigences d'une tâche avec les descriptions des services disponibles durant le processus de composition. En agrégeant tous les services disponibles dans un seul graphe, l'objectif est essentiellement de maintenir une base de connaissances agrégées qui peut être interrogée pour résoudre des tâches complexes. Similaire au processus d'agrégation des services, la résolution des tâches est également effectuée en deux étapes. Dans la première étape, une ou plusieurs compositions possibles sont dérivées au niveau sémantique utilisant le graphe G_p^1. Si une composition est possible au niveau sémantique, les services sous-jacents qui peuvent prendre part à la construction du nouveau service composite sont identifiés au cours de la deuxième étape utilisant le second niveau d'agrégation G_p^2. Pour chaque sommet dans le graphe *Gr* de la tâche, les correspondances sémantiques des entrées ($A = p_{in}^{semantic}$) et des sorties ($B = p_{out}^{semantic}$) des paramètres avec les nœuds du graph G_p^1 sont identifiés. Si A et B sont identifiés dans G_p^1 et s'il y a un chemin de A à B, alors il y a possibilité de fournir le service demandé. N'importe quel chemin identifié dans G_p^1 est une composition sur le plan sémantique. Les chemins retenus sont ceux ayant des paramètres syntaxiquement corrects identifiés dans G_p^2.

La composition hiérarchique de service: cette composition est basée sur le principe de l'organisation des ressources (dispositifs) en utilisant un protocole appelé LATCH. Le principe de base du processus de *latching* consiste à rattacher les dispositifs dotés de ressources limitées à des dispositifs riches en ressources afin d'effectuer leurs opérations. Ce processus est répété jusqu'à ce que le niveau 3 des dispositifs soit atteint. Le résultat du processus de *latching* est une hiérarchie dont les dispositifs de niveau 3 seront au plus haut niveau dans l'arborescence. Sur la base de la hiérarchie formée, les auteurs dans **[92]** définissent la zone de service et la zone de recherche pour un dispositif D. La zone de service d'un dispositif D comprend le dispositif D lui-même ainsi que tous ses fils dans la hiérarchie. La zone de recherche d'un dispositif D est sa propre zone de service si son niveau (α) dans la hiérarchie est supérieur à 1, soit α (D)> 1. Si α (D) <1 alors la zone de recherche de D est la zone de service du parent de D. Quand une tâche particulière doit être accomplie, sur la base du α du dispositif générant la demande, le processus de résolution des tâches commence soit par le même dispositif ou bien par son parent immédiat. Ce processus se répète jusqu'à ce que le nœud le plus élevé (par exemple, B et α (B) = 3) dans la hiérarchie soit atteint. Au niveau du B, tous les services disponibles au sein de l'environnement sont inspectés en concertation avec les autres dispositifs du même niveau (α = 3) pour arriver à une composition possible.

4.2.7.5. Modèle de monitoring de services

Quand un nouvel dispositif A rejoint la hiérarchie des dispositifs, son parent B collectera tous les graphes des services fournis par A et met à jour son graph local d'agrégation $G^B{}_P$. Un algorithme est présenté pour détailler la procédure utilisée pour intégrer un nouveau nœud dans la hiérarchie. L'information mise à jour est envoyée dans la hiérarchie vers tous les

parents de B. Quand un dispositif A quitte un parent B, soit en raison de sa mobilité ou en raison de la limitation de ses ressources, il peut en aviser B de son départ. B peut également décider que A n'est plus disponible en raison de la non-réception des messages périodiques. Une fois le dispositif A n'est plus disponible, tous les services qu'il offrait ne devrait plus être contenus dans l'agrégation de B. De même, B supprime tous les services associés à A et envoie un message de mise à jour à tous ses parents.

4.2.7.6. Modèle de communication

Le mécanisme de composition de service présenté se base sur la plate-forme middleware orientée services appelée PICO (*Pervasive Information Communities Organization*) **[289]**. Cette plate-forme offre un cadre transparent aussi bien pour les applications que pour les services afin de fonctionner et de coopérer les uns avec les autres d'une manière efficace. PICO fournit des moyens pour extraire les capacités des ressources et permet aux services d'être construit autour de ces capacités. En créant une structure de coopération entre les services conçus, la réalisation des objectifs plus complexes est alors possible. Le modèle de dispositif fourni capte les caractéristiques des ressources matérielles disponibles. Les caractéristiques identifiées sont offertes comme des services sur le réseau en créant des entités logicielles appelées *delegents* (délégués intelligents). Beaucoup de ces *delegents* peuvent être combinés ensemble dans une structure coopérative appelée communauté. Les communautés permettent une utilisation transparente des services dans la plateforme PICO.

4.2.8. SeGSeC

4.2.8.1. Architecture générale

Dans **[173]**, les auteurs proposent un mécanisme de composition appelée SeGSeC (*Semantic Graph-Based Service Composition*) pour la composition de service à partir de multiples composants en se basant sur la sémantique du service demandé par l'utilisateur. SeGSeC intègre un modèle sémantique pour la modélisation des composants appelé CoSMoS (*Component Service Model with Semantic*). SeGSeC se compose de quatre modules comme le montre la **figure 4.14**: *RequestAnalyzer, ServiceComposer, SemanticsAnalyzer,* et *ServicePerformer.*

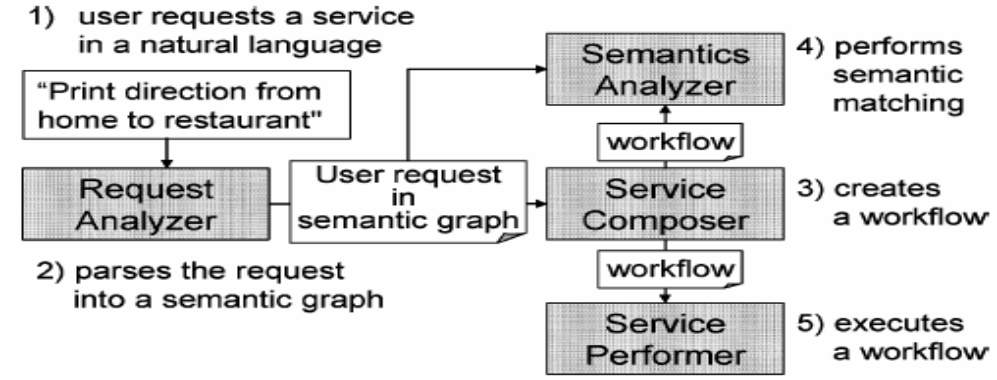


Figure 4.14 : Les modules dans SeGSeC **[173]**.

Quand un utilisateur demande un service dans un langage naturel **[Figure 4.14, (1)]**, *RequestAnalyzer* transforme cette demande du langage naturel en une représentation graphique sémantique CoSMoS et, ensuite, transmet ce graphe sémantique à

ServiceComposer **[Figure 4.14, (2)]**. A la réception de la demande de l'utilisateur, *ServiceComposer* découvre les composants nécessaires en se basant sur la demande de l'utilisateur et crée un *workflow* en utilisant les composants découverts **[Figure 4.14, (3)]**. Le *ServiceComposer* passe le *workflow* obtenu ainsi que la demande de l'utilisateur à *SemanticsAnalyzer*, qui à son tour examine si la sémantique du *workflow* répond à la demande de l'utilisateur **[Figure 4.14, (4)]**. Si *SemanticsAnalyzer* conclut que la sémantique du *workflow* répond à la demande de l'utilisateur, il notifie le résultat à *ServiceComposer*. Ce dernier passe le *workflow* examiné à *ServicePerformer* qui se charge de son exécution **[Figure 4.14, (5)]**.

4.2.8.2. Modèle des connaissances

SeGSeC utilise le modèle de composants basé sur la sémantique appelé CoSMoS. A l'image des modèles de composants existants (par exemple, WSDL, JavaBeans / EJB, COM, etc.), CoSMoS définit un composant en spécifiant ses opérations et ses propriétés. En outre, afin de modéliser la sémantique d'un composant, CoSMoS introduit les concepts comme des entités représentant des idées abstraites (par exemple, "direction" et "restaurant") et des actions (par exemple, "imprimer" et "générer"), et annote la sémantique des opérations, des entrées, des sorties et des propriétés du composant en utilisant des concepts. Par exemple, avec CoSMoS, la sémantique du composant générateur d'une direction (c'est à dire un composant pour générer une carte montrant une direction d'une adresse vers une autre) peut être modélisée en spécifiant les concepts "origine", "destination" et "direction", comme étant la sémantique des deux adresses en entrée et l'image JPEG montrant la direction en sortie, respectivement. CoSMoS utilise également un concept pour préciser la relation entre deux concepts. Par exemple, le concept "de" peut être utilisé pour spécifier la relation entre les concepts "origine" et "direction", pour spécifier que la " direction" est "de" l' "origine". Bien que cette approche de modélisation de la sémantique des composants utilisant des concepts soit similaire à l'approche proposée par les standards de services Web sémantiques, tels qu'OWL-S **[85]** et WSMO **[104]**, CoSMoS supporte l'annotation du concept d'une opération d'un composant en plus des concepts d'une entrée, d'une sortie, et d'une propriété d'un composant. Afin de définir un composant en utilisant des opérations, des propriétés et des concepts dans un format compréhensible par une machine, CoSMoS décrit un composant sous forme d'un graphe sémantique qui se compose de nœuds et de liaisons marquées. Les nœuds dans le graphe sémantique représentent les opérations, les entrées, les sorties et les propriétés d'un composant, ainsi que leurs types de données et leurs concepts. Les liens marqués dans le graphe sémantique représentent les relations entre les nœuds. Afin de décrire un composant comme un graphe sémantique, CoSMoS définit des classes qui sont utilisées pour préciser ce que représentent chaque nœud et chaque lien marqué dans un graphe sémantique. En fonction de leurs rôles, il existe quatre domaines pour les classes définies dans CoSMoS, à savoir le domaine des composants, le domaine de type de données, le domaine des sémantiques, et le domaine de logique.

4.2.8.3. Modèle de *matching* de services

Après avoir reçu un *workflow* et la demande de l'utilisateur du *ServiceComposer*, *SemanticsAnalyzer* effectue une étape appelée *matching* sémantique et examine si la sémantique du *workflow* répond à la demande de l'utilisateur ou non. *SemanticsAnalyzer*

examine le *workflow* par: 1) la conversion de ce *workflow* en un graphe sémantique; 2) l'ajout de nouveaux liens au graphe sémantique résultant afin qu'il modélise la sémantique du *workflow*, et 3) la vérification si tous les liens de la demande de l'utilisateur apparaissent également dans le graphe sémantique obtenu (i.e., le graphe sémantique qui modélise la sémantique du *workflow*).

4.2.8.4. Modèle de sélection de services

Le mécanisme de sélection est effectuée par le *ServiceComposer* afin de déterminer quels composants (i.e., composants dont les sorties ou les propriétés sont compatibles avec les entrées de l'opération donnée) à ajouter au *workflow* parmi ceux déjà découverts. Ce mécanisme considère tout d'abord toutes les combinaisons possibles des sorties, des propriétés et des valeurs littérales des composants découverts, tel que chacune des combinaisons fournit toutes les entrées de l'opération donnée. Ensuite, il calcule la valeur de similarité de chaque combinaison. La valeur de similarité d'une combinaison est calculée comme le nombre des concepts qui apparaissent à la fois dans la demande de l'utilisateur et dans les représentations des graphes sémantiques des composants qui fournissent les sorties et les propriétés contenues dans la combinaison. Enfin, le mécanisme de sélection choisit la combinaison qui offre la valeur de similarité la plus forte. Selon la combinaison sélectionnée, *ServiceComposer* créera différents *workflows.*

4.2.8.5. Modèle de composition de services

Lorsqu'un utilisateur demande un service dans un langage naturel (par exemple, "imprimer la direction de la maison au restaurant"), *RequestAnalyzer* transforme d'abord la requête en une représentation graphe sémantique de CoSMoS. SeGSeC exige que la requête contienne un prédicat (par exemple, "imprimer"). SeGSeC suppose que *RequestAnalyzer* utilise des techniques existantes (comme celles données dans **[290]**) pour transformer une phrase en langage naturel dans un graphe sémantique. Après l'analyse d'une phrase décrite dans un langage naturel en un graphe sémantique, *RequestAnalyzer* passe le graphe sémantique de la demande de l'utilisateur au *ServiceComposer*. Ce dernier découvre alors les composants nécessaires en se basant sur la demande de l'utilisateur et crée un *workflow* en utilisant les composants découverts. Afin de créer un *workflow* pour la demande de l'utilisateur, *ServiceComposer* découvre d'abord un composant initial dont le fonctionnement effectue le prédicat spécifié dans la demande de l'utilisateur, et crée un *workflow* qui contient uniquement ce composant initial. Ensuite, *ServiceComposer* découvre les autres composants qui fournissent les entrées de l'opération du composant initial, et étend le *workflow* initial en ajoutant ces composants. Cette étape est appelée complément d'entrée (*Input Complement*). Le complément d'entrée est implémenté comme une fonction récursive avec trois arguments : une opération dont les entrées doivent être découvertes, une demande de l'utilisateur, et un *workflow* à élargir par la fonction. *ServiceComposer* initie les arguments de cette fonction en fournissant l'opération du composant initial, la demande de l'utilisateur, et le *workflow* contenant le composant initial.

4.2.8.6. Modèle de monitoring de services

Le monitoring des services est réalisé dans deux cas: (1) lorsque le *ServiceComposer* ne parvient pas à découvrir un composant dont le fonctionnement assure le prédicat spécifié, il

essaie de découvrir un autre composant à l'aide d'un prédicat qui est compatible avec le prédicat spécifié. Si *ServiceComposer* n'arrive toujours pas à découvrir aucun composant, il avertit l'utilisateur qu'il ne peut pas composer le service demandé ; et (2) lorsque *SemanticsAnalyzer* examine la sémantique du *workflow* qui est reçu du *ServiceComposer*, s'il conclut que la sémantique de ce *workflow* ne satisfait pas la demande de l'utilisateur, alors *ServiceComposer* essaie de créer d'autres *workflows* sur la base de la requête utilisateur.

4.2.8.7. Modèle d'évaluation

L'approche SeGSeC est illustrée par un exemple de scénario qui montre la façon dont le service d'impression d'une direction est composé quand un utilisateur demande le service en fournissant une phrase: "imprimer la direction de la maison au restaurant". Initialement, quatre composants sont disponibles : Accueil, Restaurant, Générateur de Direction, et Imprimante. Supposons que l'utilisateur veut emmener sa famille dans un nouveau restaurant qu'il a récemment trouvé sur le web, donc il veut imprimer une carte montrant la direction au restaurant depuis sa maison. Supposons que: 1) le serveur Web du restaurant stocke les informations du restaurant tel que son adresse dans un document structuré (par exemple, en XML); 2) le PC de l'utilisateur stocke ses informations personnelles, telle que son adresse de résidence dans une base de données ; 3) il y a un service Web qui, étant donné deux adresses, génère une carte montrant une direction allant d'une adresse vers l'autre ; et 4) l'utilisateur a une imprimante connectée à son réseau à la maison. Afin d'imprimer la carte montrant la direction de son domicile vers le restaurant, l'utilisateur n'a pas besoin de savoir comment faire entrer l'adresse du restaurant au service Web qui génère une carte. La composition dynamique de service composera automatiquement le service d'impression de la direction, à la demande de l'utilisateur, en découvrant les quatre composants nécessaires. Puisque le service d'impression de la direction est composé juste à la demande de l'utilisateur, il n'est pas nécessaire de le configurer ou de le déployer à l'avance.

4.2.8.8. Modèle de communication

SeGSeC se base sur un middleware appelé CoRE (*Component Runtime Environment*) montré sur la **figure 4.15**, et ce, afin de découvrir et d'exécuter les composants. CoRE est conçu pour supporter CoSMoS sur diverses technologies de composants. Il se compose de deux interfaces, interface de découverte et interface d'accès, et trois groupes de modules, à savoir *DiscoveryEngine, InvokerEngine* et *PropertyAccessEngine*.

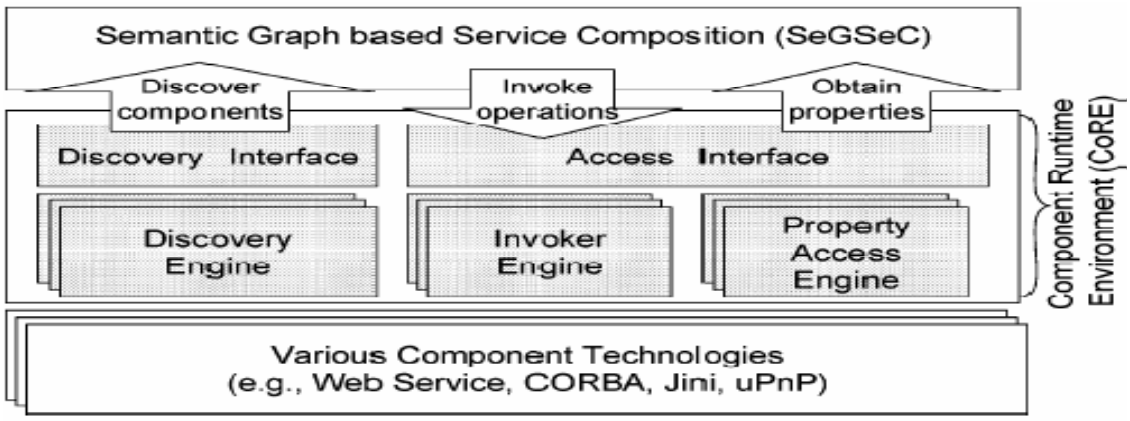


Figure 4.15 : L'architecture de CoRE **[173]**.

L'interface de découverte fournit une interface pour découvrir un composant distribué dans un réseau. L'interface d'accès fournit une interface pour invoquer une opération d'un composant et pour récupérer une propriété d'un composant. Sur réception d'une requête à partir de l'interface de découverte, le *DiscoveryEngine* découvre un ou plusieurs composants et fournit une représentation en graphe sémantique CoSMoS des composants découverts (s) (par exemple, en analysant les métadonnées du composant décrit dans CoSMoS / XML). Sur réception d'une requête à partir de l'interface d'accès, *InvokerEngine* invoque une opération d'un composant, et le *PropertyAccessEngine* récupère une propriété d'un composant.

CoRE a été implémenté en Java **[291]**. Cette implémentation comprend un *DiscoveryEngine* pour découvrir les composants stockés sur un hôte local, un *InvokerEngine* pour invoquer les méthodes des composants, et un *PropertyAccessEnvine* pour récupérer les propriétés des composants. L'implémentation de base reposant sur les technologies des services Web est également en cours de développement.

4.2.9. URS

4.2.9.1. Architecture générale

URS (*Ubiquitous Robotic Space*) **[170]** se réfère à un environnement particulier dans lequel les robots peuvent répondre intelligemment aux besoins des utilisateurs selon le contexte courant de l'espace en se basant sur les capteurs et les actionneurs distribués dans l'environnement. Son principal objectif est d'intégrer un grand nombre de technologies hétérogènes avec des composants robotiques afin d'implémenter des services robotiques réels. La **figure 4.16** illustre la structure conceptuelle du projet URS, qui comprend trois espaces à savoir le physique, la sémantique et le virtuel. Les auteurs décrivent dans **[170]** la façon d'élaborer les éléments fondamentaux de chaque espace et la façon d'intégrer les trois espaces développés de manière indépendante dans une application robotique réelle.

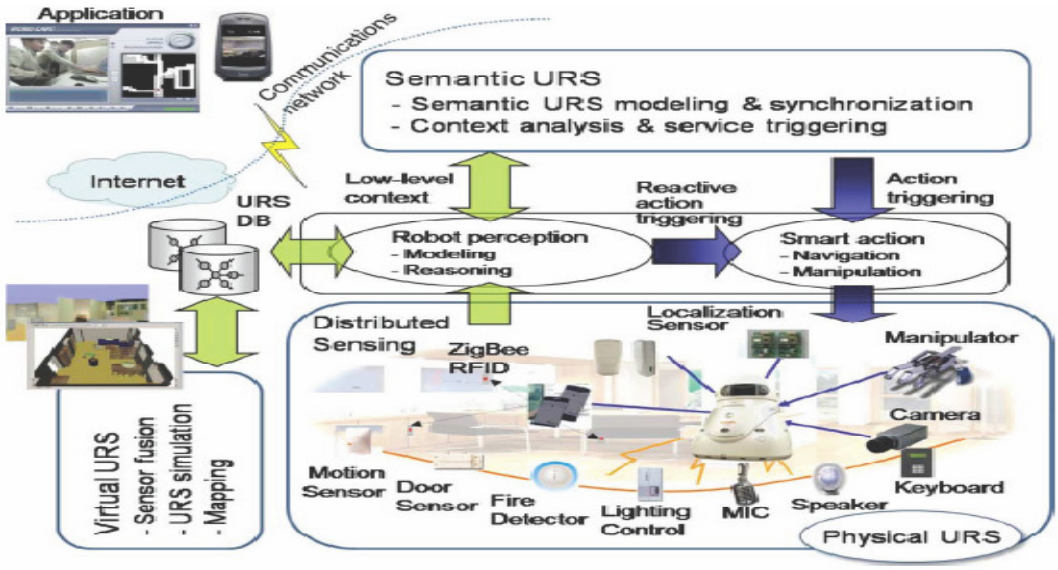


Figure 4.16 : La structure conceptuelle de l'espace robotique ubiquitaire **[170]**.

L'espace physique est supporté par deux éléments fondamentaux: les réseaux de capteurs sans fil et les réseaux de localisation afin de contrôler le comportement d'un robot à l'intérieur de l'espace physique. Un autre élément essentiel pour la gestion des trois espaces et les flux de

données entre eux est le *serveur URS*. Ce serveur s'occupe également de la gestion de la base de données pour les multi-utilisateurs, les multi-robots, et les changements dynamiques dans l'URS. A cela s'ajoute la gestion des différentes connectivités entre les utilisateurs et Internet ou d'autres réseaux de communication existants. L'espace sémantique est constitué de deux modules principaux: l'exécution de service sensible au contexte (*context-aware service execution : CASE*) et la reconfiguration dynamique (*dynamic reconfiguration : DR*). CASE interprète les données collectées à partir des nœuds du réseau de capteurs *u-Clips* afin de comprendre la situation actuelle et générer des plans d'action prenant en charge cette situation. DR gère le comportement dynamique de l'URS en ajoutant ou en supprimant les services URS en réponse à des changements dynamiques dans l'espace physique, tels que le déploiement ou le retrait de dispositifs. Le rôle principal de l'espace virtuel est de fournir à l'utilisateur un modèle virtuel 2-D ou 3-D de l'espace physique, permettant ainsi à cet utilisateur d'étudier et d'interagir avec l'espace physique d'une manière intuitive. Dans l'approche proposée, les formats suivants : *Scalable Vector Graphics* (SVGs) et *Virtual Reality Modeling Language* (VRML) sont utilisés pour modéliser les cartes 2-D et 3-D, respectivement.

4.2.9.2. Modèle des connaissances

Le modèle de connaissance proposé comprend trois parties principales, à savoir la carte de l'espace sémantique (*Semantic Space Map*), les descriptions d'une situation (*Situation Descriptions*) et les connaissances sur le service d'un robot (*Robot Service Knowledge*). Le modèle de la carte de l'espace sémantique est basé sur une ontologie OWL afin de décrire la configuration structurelle et sémantique de l'espace physique. Ce modèle décrit une position ou une région physique spécifique en utilisant des symboles et des phrases formelles. Le modèle des descriptions d'une situation représente l'état dynamique de l'espace physique. Une description d'une situation est représentée comme une paire contenant un descripteur de contexte et sa valeur. Chaque description d'une situation est envoyée pour le stockage au niveau d'un tableau de la situation (*situation board*). Le tableau d'une situation est le point de référence pour toutes les descriptions de la situation concernant l'état actuel de l'URS. Le modèle des descriptions d'une situation est basé sur trois éléments différents: les capteurs, les robots et les nœuds d'interprétation du contexte (*context interpretation* : *ci*). Chaque capteur est modélisé comme une opération avec des sorties seulement, et les connaissances sur le service du robot et les nœuds *ci* sont modélisés comme des opérations avec des entrées et des sorties. Les connaissances sur le service d'un robot comprennent les règles et les ontologies nécessaires pour détecter des situations spécifiques à une application et déclencher des commandes d'actions appropriées pour chaque situation.

4.2.9.3. Modèle de composition de services

Le modèle de composition de service est basé sur le réseau d'interprétation du contexte (*Context Interpretation Network : CIN*). Un CIN est une chaîne construite à partir des capteurs, des robots et des nœuds *ci* comme un service composite pour l'interprétation du contexte. Le CIN est construit dynamiquement par le module DR au moment de l'installation ou de la suppression des dispositifs capteurs du réseau de capteurs *u-Clips*. La planification *Means-end* **[292]** est adoptée pour construire une chaîne CIN et les connaissances sur le service du robot en faisant correspondre l'entrée et la sortie de chacune des opérations modélisées. La **figure 4.17** montre un exemple d'une chaîne d'un service composite. Dans

cette chaîne, si un dispositif nouvellement déployé est identifié comme un service *TemperatureSensor* ayant une sortie *Temperature*, alors il sera lié à l'entrée de l'interpréteur *WarmthInterpreter.*

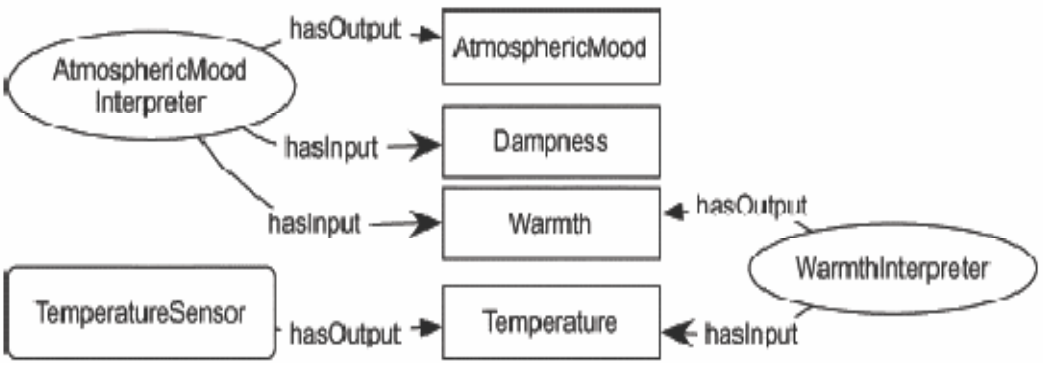


Figure 4.17 : Un exemple d'une chaîne CIN dans l'espace sémantique **[170].**

4.2.9.4. Modèle de monitoring de services

L'URS adopte une approche centrée sur la reconfiguration des services basée sur le profil des dispositifs. Chaque dispositif dans l'espace physique envoie périodiquement son paquet d'annonce, appelé *device announcement packet* (DAP), contenant son identifiant unique (ID). Le module DR reçoit d'abord les DAPs transmis par chaque dispositif, ensuite il extrait les IDs et récupère les profils de tous les dispositifs afin de construire une carte reflétant leurs capacités. Enfin, DR reconfigure le service composite en ajoutant ou en supprimant des nœuds *ci*, ou des connaissances du service robot, à chaque fois que des changements sont détectés sur la carte des capacités des dispositifs.

4.2.9.5. Modèle d'évaluation

Pour décrire la façon d'appliquer l'URS sur un environnement réel, les auteurs dans **[170]** ont développé un prototype d'un service robotique de sécurité pour la surveillance d'un bureau. L'URS proposé est implémenté au rez-de-chaussée d'un immeuble, dont la superficie mesure 23,11m x 25,52m. La zone de sécurité du robot est divisée en deux sous-zones en tenant compte de la distance de transmission radio des points d'accès (commutateurs) employés et la structure spatiale de l'environnement. Chaque sous-zone a son propre système de coordonnées cartésiennes; les relations entre les systèmes de coordonnées sont stockées dans le serveur URS. Le robot joue le rôle de passerelle pour recueillir les données sur l'environnement fournies par le réseau de capteurs *u-Clips* et les données de localisation fournies par le réseau de localisation, puis transfère ces données au serveur URS via une connexion sans fil à Internet. L'espace sémantique est chargé de surveiller l'espace physique et le contrôle du robot selon le scénario de service. Dans ce scénario, des situations irrégulières, telles que les intrusions ou les niveaux faibles de batterie du robot, sont détectés à partir du traitement de l'information contextuelle à l'aide du moteur *eBossam* **[293]**. L'espace sémantique assigne une nouvelle tâche de navigation au robot lors de la détection d'une irrégularité (intrusion). De plus, la situation irrégulière est signalée à l'utilisateur sous forme d'un SMS.

4.2.9.6. Modèle de communication

Les fonctions de l'espace sémantique, incluant le traitement du modèle de l'espace, la représentation des connaissances du service robot et la découverte des services par la planification *means-end* sont toutes implémentées en utilisant un moteur de règles intégré, appelé *eBossam* **[293]**. *eBossam*, développé en langage C++, est un moteur de règles de production en chainage-avant (*forward-chaining*) construites sur la base de l'algorithme *RETE* **[294]**. La plupart des fonctionnalités de l'espace sémantique sont installées sur une carte *ARMbased* incorporée à l'intérieur d'un robot. Le référentiel des profils des dispositifs et le référentiel des services sont déployés comme des applications en *Java servlet* dans le serveur URS pour rendre les services accessibles publiquement, comme des applications Web. En outre, l'espace sémantique utilise des sockets TCP simples pour interfacer avec l'espace physique, qui comprend un robot et un réseau de capteurs appelé *u-Clips.* En outre, des protocoles de communication des données sont définis entre les trois espaces. Un message pour échanger des données entre les trois espaces se compose d'un en-tête du message sur six octets et d'une zone de données de longueur variable.

4.2.10. VRESCo

4.2.10.1. Architecture générale

L'objectif principal de VReSCO (*Vienna Runtime Environment for Service-oriented Computing*) **[172]** consiste à fournir une infrastructure (via une API côté client) aux développeurs d'applications afin de développer d'une manière efficace et flexible des applications orientées service. En outre, cette infrastructure permet de remédier à de nombreuses lacunes des Frameworks existants, à savoir le manque de dynamisme requis pour l'implémentation des architectures SOA **[295, 296]**. La bibliothèque côté client gère de façon transparente la communication SOAP avec l'infrastructure VReSCO et ses services offerts.

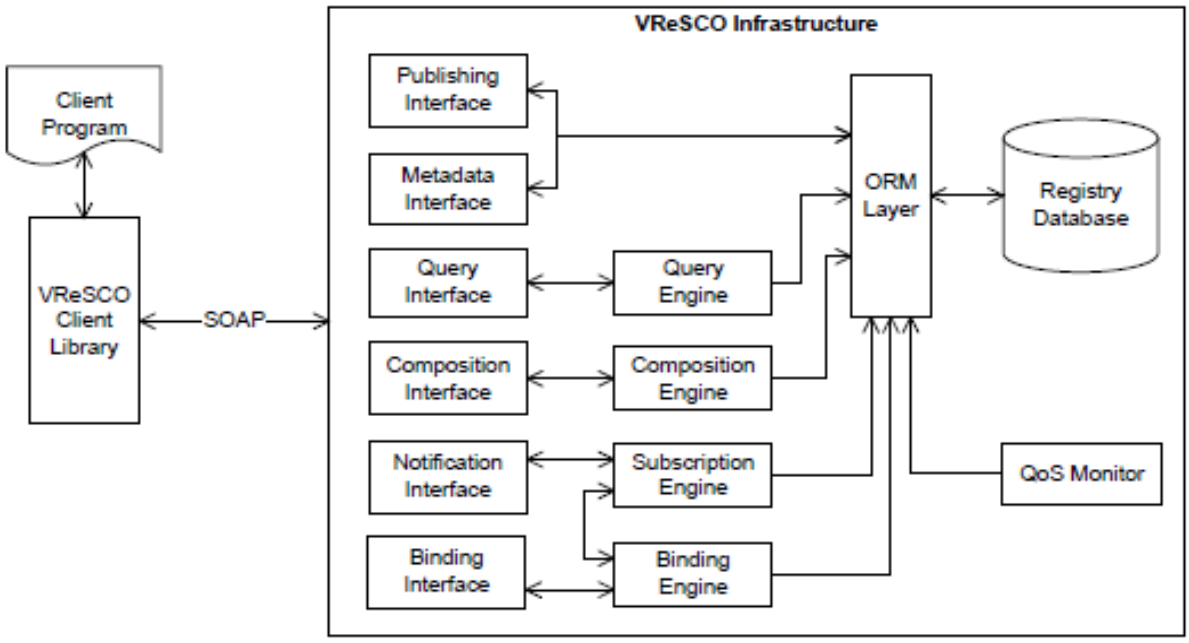


Figure 4.18 : L'architecture de VReSCO **[295]**.

Comme le montre la **figure 4.18**, l'architecture globale de VReSCO fournit plusieurs services au développeur d'applications, à savoir le service des publications et des métadonnées, le service de recherche et d'interrogation, le service de liaison et d'invocation, le service de notification, et le service de composition. Le processus de composition de services suppose que les exigences du service composite en termes de fonctionnalités sont déjà définies. La composition de services comprend les étapes suivantes : la découverte des services, la spécification du service composite et son déploiement, les tests et le monitoring de l'exécution du service composite. L'approche de composition en tant que service (*Composition as a Service: CaaS*) est combinée avec le module d'exécution de VReSCO pour englober toutes ces étapes (sauf les tests) du processus de composition dans une approche holistique. L'approche CaaS repose sur l'idée de réduire la complexité du développement d'un service composite. VRESCO fournit un mécanisme permettant de publier des services dans son propre registre et les associer à des métadonnées expressives qui peuvent ensuite être exploitées par un langage d'interrogation.

4.2.10.2. Modèle des connaissances

Le modèle de connaissances est basé sur le langage VCL (*Vienna Composition Language*). Ce langage permet de spécifier les exigences d'une composition en termes de la *QoS* globale nécessaire ainsi que la *QoS* locale requise des différents services constituant cette composition. En outre, des contraintes sur les services individuels peuvent être imposées, par exemple sur les entrées et les sorties de chaque service, ainsi que les pré-conditions ou les post-conditions qui doivent être satisfaites. Les contraintes de la *QoS* sont regroupées en quatre contraintes hiérarchiques et une valeur de préférence est associée à chaque contrainte exprimant son importance dans la résolution de contraintes. VCL utilise quatre valeurs hiérarchiques différentes par défaut (*requis, forte, moyenne, faible*) qui peuvent être spécifiées pour chaque contrainte de la *QoS* globale ou locale qui sera respectée par l'algorithme de composition. Dans ce modèle de connaissances, un service est défini par ses caractéristiques fonctionnelles et non-fonctionnelles **[297]**. Le modèle de métadonnées de VReSCO décrit la fonctionnalité d'un service par sa catégorie, sa fonction, ses pré-conditions et ses post-conditions. Outre les caractéristiques fonctionnelles, un ensemble d'attributs non-fonctionnels de la *QoS* est associé à chaque service. Deux types d'attributs de la *QoS* sont distingués : déterministe et non-déterministe. Les attributs déterministes englobent les attributs dont les valeurs sont connues avant l'invocation d'un service, tandis que les attributs non-déterministes ne sont pas connus au moment de l'invocation. Par conséquent, ces attributs doivent être captés et calculés par l'observation de l'exécution.

4.2.10.3. Modèle d'évaluation de la QoS et du contexte

Les attributs de qualité de service sont gérés et mis à disposition de l'infrastructure VReSCO grâce à l'approche inspirée de **[298]**. Les auteurs dans **[172]** définissent trois attributs déterministes (*prix, messagerie fiable, sécurité*) et cinq attributs non-déterministes (*temps de réponse, latence, disponibilité, précision, débit*). Pour chaque attribut de la *QoS* considéré, une formule de calcul de l'attribut est fournie dans les cas déterministe et non-déterministe. Le *temps de réponse* est la durée d'attente d'une demande de service en plus du temps d'exécution de ce service au niveau de son fournisseur. Il est calculé comme la valeur moyenne de *n* points de mesure individuels. *La latence* est la durée d'attente d'une demande

de service et sa moyenne est calculée de la même manière que le temps de réponse. *La disponibilité* est la probabilité qu'un service fonctionne et produit des résultats corrects. *La précision* est la probabilité qu'un service produit des résultats corrects. *Le débit* est le nombre maximum de demandes qu'un service peut traiter dans un certain laps de temps.

4.2.10.4. Modèle de sélection de services

La sélection des services est basée sur le problème de satisfaction de contraintes (CSP), qui prend en entrée l'ensemble des services candidats regroupés par fonction. Ces services sont fournis par le processus de résolution de fonctionnalité (*feature resolution process*). Les contraintes locales (seulement celles qui sont optionnelles) et globales (celles obligatoires et optionnelles) spécifiées dans VCL seront ajoutées comme des contraintes CSP. Le graphe de composition structuré est utilisé pour agréger toutes les contraintes globales de la *QoS* en se basant sur leurs formules d'agrégation et le flux de contrôle de la composition. Puisqu'un CSP peut avoir plusieurs solutions, une pondération est alors associée à chaque niveau hiérarchique des contraintes en vue de trouver une solution optimale.

4.2.10.5. Modèle de composition de services

La résolution des fonctionnalités (*Feature resolution*) est le processus d'interrogation de tous les services candidats qui implémentent une fonction donnée avec ses contraintes spécifiées dans VCL. En plus du nom de la fonction, toutes les contraintes de la *QoS* requise, entrée, sortie, pré et post-conditions sont ajoutés comme critères à la requête afin d'assurer que la fonction répond à toutes ces contraintes. La génération de la structure de la composition (*Structure Composition Generation*) se passe en parallèle avec la résolution des fonctionnalités car il n'y a pas de dépendances entre les deux processus. Le service de composition analyse la spécification du protocole décrit en VCL et spécifie l'ordre d'exécution des services. L'objectif principal de cette étape est d'analyser les dépendances de données entre les invocations de services et de générer une composition de service structurée (i.e. déterminer l'ordre d'exécution correcte). Dans ce modèle de composition, l'approche décrite dans **[299]** est adoptée et étendue afin de générer une composition structurée et également pour calculer la qualité de service globale d'une composition.

4.2.10.6. Modèle d'évaluation

Les auteurs dans **[172]** décrivent un exemple simple de composition d'un service de "guide restaurant". Ce service composite peut être utilisé sur des appareils mobiles afin de localiser rapidement et réserver un restaurant pour le jour même, à un moment précis et pour un certain nombre de personnes. Tous les services impliqués dans la composition sont supposés déjà publiés dans le registre VReSCO et associés à des métadonnées. La composition utilise quatre fonctions définies dans le registre VReSCO, et plusieurs services sont associés à ces fonctions (par exemple, certains services de restauration, les services des taxis locaux et les services pour cartes de direction, tels que Google Maps, Yahoo, etc.). Ces services sont transformés à des fonctions en utilisant la bibliothèque cliente de VReSCO. Après la définition des fonctionnalités, un certain nombre de contraintes sur les fonctions individuelles sont définies. Les contraintes globales définissent l'interface externe du service composite en utilisant la contrainte sur l'entrée et la sortie. La contrainte de la *QoS* définit quelques propriétés générales de qualité de service qui doivent être satisfaites- en fonction de

leur niveau hiérarchique - avant que la composition soit générée et déployée. Cet exemple définit uniquement le temps de réponse comme contrainte exigeante. Tous les autres paramètres sont optionnels (*nice to have*) avec une préférence pour le prix sur la sécurité.

4.2.11. WComp

4.2.11.1. Architecture générale

Le middleware WComp (*Middleware for Ubiquitous Computing*) **[162]** fédère trois principaux paradigmes: le paradigme des services Web basés sur les événements (*Event-based web services paradigm*), le paradigme basé sur des composants légers à l'intérieur des services Web composites (*Lightweight component-based paradigm inside composite web services*), et le paradigme d'adaptation utilisant le concept original appelé Aspect d'Assemblage (*Aspect of Assembly : AA*).

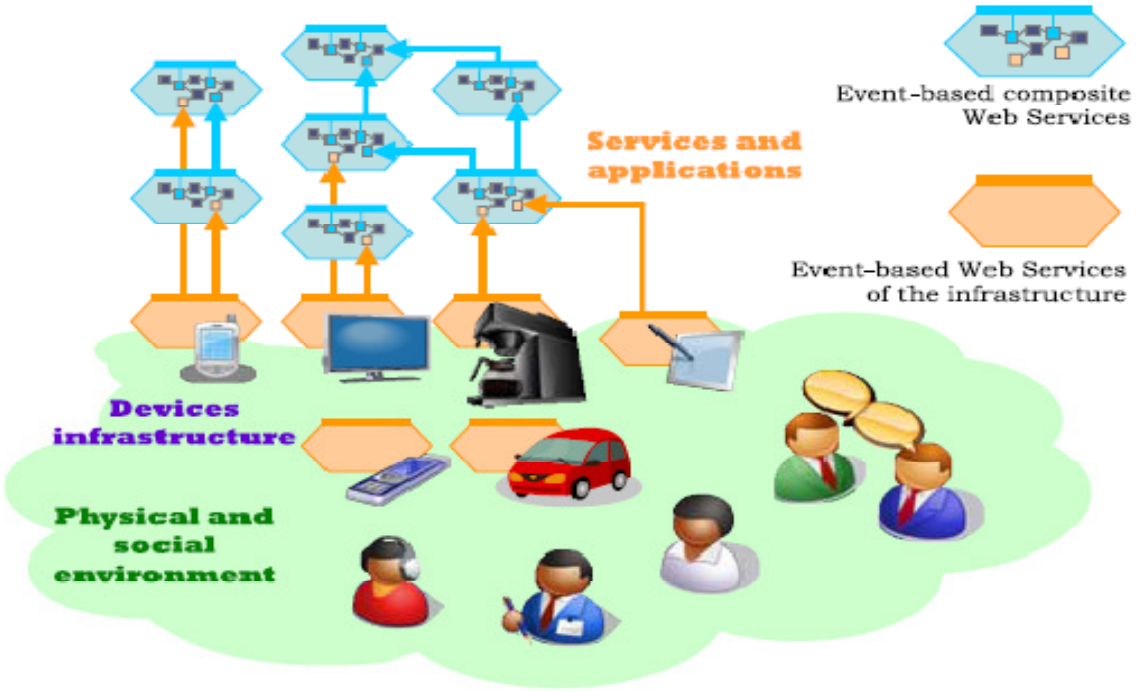


Figure 4.19 : Un graphe des services Web basés sur les événements **[162]**.

Le paradigme des services Web basés sur les événements stipule qu'une application ubiquitaire est construite comme un graphe de services Web basés sur les événements. Deux types de services sont distingués: les services composites et les services basiques. Les services composites sont des services dont l'exécution appelle d'autres services, et ce, contrairement aux services basiques, dont les implémentations sont autonomes et n'invoquent pas d'autres services. Les services basiques sont généralement des services Web pour dispositifs (*Web Services for Devices*) comme UPnP ou DPWS. Le paradigme basé sur des composants légers à l'intérieur des services Web composites utilise l'architecture d'un composant léger de service (*Service Lightweight Component Architecture : SLCA*) **[127]**. Les services Web composites sont créés à partir d'un assemblage dynamique de composants vus comme des boîtes noires. Un service composite s'exécute dans un conteneur WComp local qui gère un ensemble dynamique de composants WComp légers et qui fournit une interface d'un service Web basé sur les événements (**Figure 4.19**). Le paradigme d'adaptation utilisant le concept

original d'Aspect d'Assemblage (AA) permet la préparation des schémas types indépendants et transversaux portant sur l'adaptation des services Web composites. Ces schémas sont logiquement fusionnables en cas de conflits et applicables à tous les services Web composites de la l'application, pas nécessairement connus à priori. Les Aspects d'Assemblage (AA) fournissent une adaptation structurelle au modèle de composition, puisque l'assemblage des composants internes d'un service composite est modifié, sans modifier ses composants (boîtes noires) de base.

4.2.11.2. Modèle des connaissances

Le modèle des connaissances est basé à la fois sur l'architectures d'un composant léger de service (SLCA) et les aspects d'assemblage (AA). Le méta-modèle SLCA est basé sur des composants légers LCA (*Lightweight Component Architecture*) pour concevoir la composition des services Web. Un service composite encapsule le conteneur SLCA qui contient un assemblage de composants dynamiques et légers. Le modèle de composant LCA est un modèle dérivé à partir de Java Beans, adapté à d'autres langages de programmation, avec des concepts des ports d'entrée/sortie et des propriétés. Le méta-modèle SLCA permet à la fois l'ajout de nouveaux services à l'environnement et l'utilisation des services dans un service composite. Un aspect d'assemblage est structuré comme un aspect contenant un point de coupure (*pointcut)* et une instruction d'adaptation (*adaptation advice*). Un aspect est spécifié dans un langage DSL (*domain-specific language*) en utilisant la spécification d'interaction initialement définie dans **[300]**, puis améliorée dans **[301]** par l'intégration des déclarations orientées événement. Avec l'approche basée sur les AAs, les développeurs de logiciels *pervasifs* et auto-adaptatifs peuvent raisonner, planifier et valider les assemblages à tous les niveaux de la phase de développement. En utilisant des règles de validation logiques prédéfinies, les incompatibilités de configurations logiques peuvent être détectées lors de l'exécution.

4.2.11.3. Modèle de composition de services

Deux approches de composition sont fournies par le middleware WComp afin de concevoir des applications ubiquitaires dynamiques: la composition pour des services de plus haut niveau (*composition for higher-level services*) et la composition pour l'adaptation (*composition for adaptation*). La composition pour des services de plus haut niveau implémente une approche classique de composition à base de composants, en utilisant SLCA. Ce dernier définit une architecture de services Web composites basés sur les événements, qui sont construits par assemblage de composants légers. Un service composite contient alors un assemblage de composants, dans un conteneur WComp. Cette première approche conçoit d'abord des services Web composites de plus haut niveau, puis elle les intègre dans le graphe de coopération des services des applications. Elle est bien adaptée à la conception des applications dans un contexte connu. L'autre approche de composition utilise l'adaptation à l'aide des aspects d'assemblage. Cette approche est particulièrement bien adaptée pour ajuster un ensemble de services Web basés sur les événements en réaction à une variation spécifique du contexte ou même à des nouvelles préférences des utilisateurs. Dans cette seconde approche, les aspects d'assemblage sont composés par un lissage (*weaver*) selon des règles logiques de fusion avec une spécification de haut niveau. Le résultat du lissage est projeté en termes de modifications élémentaires pures (*pure elementary modifications: PEMs*) : ajout et

suppression des composants, liaison et dissociation des ports. Les composants résultants sont ensuite impliqués dans des modèles d'interaction différents.

4.2.11.4. Modèle de monitoring de services

Dans l'approche de composition basée sur SLCA, l'utilisateur est indispensable pour créer sa propre application (service composite) à partir des services présents dans l'environnement par assemblage des composants légers. Dans l'approche basée sur les AAs, un aspect d'assemblage peut être sélectionné soit par l'utilisateur ou déclenché par des changements de contexte dans un processus auto-adaptatif. En conséquence, deux types d'auto-adaptation utilisant les AAs sont définis: l'adaptation dirigée par l'utilisateur (*user-driven adaptation*) et l'adaptation dirigée par le contexte (*context-driven adaptation*). Dans l'adaptation dirigée par l'utilisateur, ce dernier peut (dé-) sélectionner un AA afin d'intégrer ou d'effacer certains comportements et fonctionnalités dans le système. Deux types d'utilisateurs sont ainsi distingués: les utilisateurs experts et les utilisateurs finaux. Les experts peuvent concevoir de nouveaux AA pour des situations nouvelles, tandis que les utilisateurs finaux n'ont pas à créer des AA, mais peuvent seulement (dé-) sélectionner les AA prédéfinis. L'adaptation dirigée par le contexte est déclenchée lorsqu'un dispositif rejoint ou quitte l'environnement du système. Dans le premier cas, le nouveau composant logiciel représentant le nouveau dispositif est ajouté à la base d'assemblage. Dans le second cas, le composant logiciel représentant le dispositif partant est dissocié et retiré de la base d'assemblage.

4.2.11.5. Modèle d'évaluation

L'approche proposée est illustrée par une application ubiquitaire multi-dispositifs pour envoyer des messages textuels via un réseau. Il y a trois modes de connexion : Wifi, GSM et, lorsqu'il n'y a plus de connexion, l'application peut stocker les messages et les envoyer à un système de cache. Cette application est développée avec des composants basiques ou des services et l'adaptation est réalisée par les aspects d'assemblage afin de réorganiser les connexions entre les entités, d'instancier de nouveaux composants logiciels, ou d'interagir avec de nouveaux services. Les aspects d'assemblage (AA) sont regroupés en trois catégories: les fonctionnalités de base, la politique énergétique et l'ajout de nouvelles fonctionnalités. Tous ces aspects d'assemblage sont appliqués à l'application initiale définie pour générer dynamiquement les connexions entre les composants ou les services afin de donner le bon comportement à l'application. Les aspects d'assemblage de "Fonctionnalités de base" sont sélectionnés exclusivement par le contexte (un utilisateur ne peut pas décider d'utiliser une communication Wifi si elle n'est pas présente dans l'environnement), mais tous les autres AAs peuvent être activés par l'utilisateur ou par le contexte de l'application. Dans le premier exemple du scénario, l'application envoie tous les messages via la connexion Wifi. Lorsque l'utilisateur veut minimiser la consommation d'énergie afin de maximiser l'autonomie (adaptation dirigée par l'utilisateur), ou lorsque la batterie est faible (adaptation dirigée par le contexte), le AA défini dans la "politique énergétique" modifie l'application afin d'envoyer toutes les données au composant cache. Le deuxième exemple du scénario montre l'adaptation de l'application fournie par un AA pour l'ajout de la "nouvelle fonction vocale". L'adaptation consiste à ajouter de nouveaux composants logiciels et de modifier les connexions entre les composants et les services pour ajouter cette nouvelle fonctionnalité. Toutes les modifications concernent la structure de l'assemblage (modifications structurelles).

4.3. Analyse et comparaison des middlewares de composition de services

4.3.1. Récapitulatif des middlewares de composition de services

Modèles / Middlewares	Architecture générale	Modèle de connaissances	Modèle d'évaluation de la QoS et du contexte	Modèle de *matching* de services	Modèle de sélection de services	Modèle de découverte de services	Modèle de composition de services	Modèle de monitoring de services	Modèle d'évaluation	Modèle de communication
COCOA-PERSE	Composée de deux couches: la couche middleware de communication et la couche middleware sémantique orientée services (SOM).	Langage COCOA-L basé sur OWL-S.	Deux types d'estimation de la QoS : estimation probabiliste basée sur l'historique et estimation pessimiste.	Distance sémantique entre deux concepts appartenant à une ontologie.	Utilise des expressions régulières extraites de l'automate de la tâche et des règles de réduction pour calculer la QoS d'un Workflow.	Réalisé par COCOA-SD qui cherche la correspondance entre les capacités des services annoncés et celles des services requis.	Réalisé par COCOA-CI qui intègre les conversations des tâches et des services modélisées par des automates à états finis (FSA).	Absent	Scenario d'assistance d'un utilisateur dans une gare de train en utilisant une application appelée *e-movie* avec un ensemble de dispositifs : PDA, écran plat large, et des serveurs de vidéos.	WSDL- SOAP
PEIS-Ecology	Un composant *PEIS* disposant d'un répertoire local de descriptions *D*, d'un composant spécial *M* pour accéder à ces descriptions et les annoncer, et éventuellement d'un *configurateur*.	Des composants *PEIS*, une écologie de PEIS (PEIS-*Ecology*) et une ontologie partagée.	Absent	Mécanisme similaire à la primitive *Find* du Framework d'ancrage (*anchoring framework*) **[258]**	Absent	Basé sur la compatibilité entre les fonctionnalités annoncées et celles requises par les PEIS de l'écologie. La compatibilité est décidée grâce à une ontologie partagée.	La configuration se base sur une approche centralisée basée sur un plan **[260],** et une approche distribuée réactive **[261].**	Mécanismes d'inscription aux signaux d'erreur provenant des *PEIS* connectés et reconfigurat-ion en cas de panne d'un composant *PEIS*.	Deux scénarios : un aspirateur autonome dans une maison équipée de capteurs et de caméras et un robot de surveillance à l'aide d'une sensation électronique (*e-nose*).	Middleware *PEIS-kernel* et une ontology *shared PEIS-Ontology*.

URC-SURF	Constituée de trois composantes : SA (*SURF Agent*), DWS (*Device Web Service*) et EKR (*Environmental Knowledge Repository*)	OWL-S, OWL	Absent	Définit trois relations sémantiques dans la syntaxe de la logique des prédicats basée sur la sémantique formelle de RDF, RDF Schema, OWL et OWL-S.	Absent	Création des requêtes sémantiques pour la recherche, dans les bases de connaissances de l'environnement EKR, des connaissances sur la planification décrite en OWL-S.	Utilise le système SHOP2 qui se base sur la planification HTN.	Monitoring en cas de changements de la localisation de l'utilisateur	Un robot assiste un utilisateur dans une maison qui contient un ensemble de dispositifs : tag et lecteurs RFID, PC, robot, etc.	Pile de communication des services Web incluant les protocoles SOAP, et http et le standard XML
MySIM	Composé de quatre modules: translation, génération, évaluation et exécution.	Utilise les concepts d'une ontologie commune et un modèle générique de description d'un service.	Métrique qui mesure le degré d'équivalence non-fonctionnelle (*QoS*) entre deux opérations fonctionnellement équivalentes.	Quatre niveaux pour le *matching* sémantique entre deux concepts : exacte, *plug in*, subsume et échec **[274].**	Choisi entre deux services fonctionnellement équivalents utilisant la métrique de QoS.	Absent	Relation de composition syntaxique et sémantique avec équivalence d'interfaces.	Gestion des services composites en tenant compte de l'apparition et la disparition des services et	Un cas d'utilisation simple appelé *MyStudio* composé de cinq services: webcam, stockage, imprimante, impression et redimensionnement.	OSGi
MEDUSA	Fourni trois couches: interopérabilité de la communication, interopérabilité des services, centrée sur l'utilisateur.	Utilise le modèle de description de service *AmIi* (*Ambient Intelligence interoperability*) **[276]**.	Optimisation basée sur les théories de l'évolution et les algorithmes génétiques **[277]**.	Matching entre les descriptions des services disponibles et celles des services requis par la requête.	Ensemble d'interfaces permettant aux utilisateurs de choisir parmi les configurations d'application possibles ou de choisir	Recherche dans un référentiel de services des descriptions de service qui correspondent à une requête de service en utilisant la fonction de	Utilise des Interfaces physiques pour les services à base des RFID. Propose quatre interfaces pour construire une composition de services:	Absent	Un exemple illustratif: deux utilisateurs développent conjointement une application en utilisant des cartes de services et un téléphone	Middleware de communication : ubiSOAP

					directement les instances des services disponibles.	découverte fournie par le modèle *AmIi* **[93]**.	manuelle, semi-manuelle, mixte et autonome.		mobile.	
MUSIC	Fourni plusieurs modules de gestion : *Context Manager*, *QoS Manager*, *Adaptation Manager*, *Adaptation Controller, Adaptation Reasoner*, et *Configuration Executor*.	Modèle sensible à la *QoS* (*QoS-aware model*) et un modèle d'accord (*agreement template*) décrivant les propriétés statiques de la *QoS* fournie par un service.	Fonction d'utilité qui mesure la satisfaction de l'utilisateur par rapport à chaque dimension de la QoS.	Absent	Sélection des composants d'une configuration qui fournissent la meilleure utilité à l'utilisateur final.	Enregistrement dynamique des récepteurs de découverte (*discovery listeners*) pour êtres informés de la découverte de certains services.	Décrit une composition abstraite comme un ensemble de rôles qui collaborent à travers des ports. A l'exécution, une composition est décrite par des plans.	*SLA Monitoring* est responsable de la vérification de l'état des SLA et de prendre des mesures appropriées en cas de violation des accords spécifiés.	Scénario d'un utilisateur qui est en voyage pour rencontrer un ami. Les services utilisés pour planifier ce voyage : un service d'itinéraire pour le métro, un service de localisation (GPS-enabled) et un service de cartographie.	OSGi, UPnP.
SeSCO	Organisation des ressources (dispositifs) en utilisant un protocole appelé LATCH	Les services et les tâches sont décrits comme des graphes.	Absent	Matching syntaxique et sémantique.	Absent	Absent	Composition centralisée et hiérarchique de services avec mécanismes de résolution des tâches.	Supporte l'arrivée et le départ des dispositifs en utilisant le processus LATCH.	Absent	Middleware PICO (*Pervasive Information Communities Organization*).
SeGSeC	Se compose de quatre modules : *RequestAnalyzer, ServiceComposer SemanticsAnalyz-er,* et *ServicePerformer*	Modèle CoSMoS (*Component Service Model with Semantic*).	Absent	Matching sémantique.	Choisit la combinaison de services qui possède la valeur de similarité la plus forte avec la demande de l'utilisateur.	Basé sur le moteur de découverte du middleware CoRE.	Crée un workflow qui contient le composant initial et étend ce *workflow* en ajoutant des composants par un processus appelé	Lorsque la sémantique d'un *workflow* ne satisfait pas la demande de l'utilisateur, le *ServiceCom*	Scénario d'impression d'une direction de la maison au restaurant à partir de quatre composants : Accueil, Restaurant, Générateur de	Middleware CoRE (*Component Runtime Environment*).

							complément d'entrée (*Input Complement*).	*poser* essaie de créer d'autres *workflows*.	Direction, et Imprimante.	
URS	Contient trois espaces: physique, sémantique et virtuel.	Comprend trois parties: la carte de l'espace sémantique basée sur OWL, les descriptions d'une situation (*Situation Descriptions*) et les connaissanc-es sur le service d'un robot basées sur des règles et une ontologie.	Absent	Matching des entrées et sorties des operations en utilisant un moteur de règles intégré appelé *eBossam* **[293].**	Absent	Utilise le moteur de règles intégré *eBossam* **[293].**	Construit une chaîne CIN *(Context Interpretation Network)* comme un service composite pour l'interprétation du contexte en utilisant la planification *Means-end* **[292].**	Reconfigurat ion des services basée sur le profil des dispositifs qui envoient périodiquem ent un paquet d'annonce appelé DAP.	Prototype d'un service robotique de sécurité pour la surveillance d'un bureau utilisant les dispositifs suivant : robot, capteur, webcam, téléphone.	Utilise un serveur URS à base de *Java servlet* pour rendre les services accessibles comme des applications Web, des sockets TCP simples pour interfacer avec l'espace physique et un protocole de communication des données.
VRESCo	Fournit plusieurs services au développeur d'application: service des publications et des métadonnées, service de recherche et d'interrogation, service de liaison et d'invocation, service de notification, et	Langage VCL (*Vienna Composition Language*).	Pour chaque attribut de la *QoS* considéré, une formule de calcul de l'attribut est fournie dans les cas déterministe et non-déterministe.	Absent	Basée sur le problème de satisfaction de contraintes (CSP).	Fournit un mécanisme pour publier des services dans son propres registre et associe des métadonnées à ces descriptions.	L'ordre d'exécution des services spécifié dans le langage VCL.	Absent	Exemple simple d'un service appelé *RestaurantGuid* utilisé sur des appareils mobiles afin de localiser rapidement et réserver un restaurant à un moment précis et pour un certain nombre	VRESCO SOA runtime.

	service de composition.								de personnes.	
WComp	Fédère trois principaux paradigmes: les services Web basés sur les événements, les composants légers à l'intérieur des services Web composites et l'adaptation utilisant le concept original appelé Aspect d'Assemblage.	Méta-modèle SLCA (*Service Lightweight Component Architecture*) et les Aspects d'Assemblage (AA).	Absent	Matching des points de coupure *(Pointcut matching)*	Absent	Souscription et notification à base d'événements.	Composition à base de composants, en utilisant SLCA et composition pour l'adaptation utilisant AA.	Adaptation dirigée par l'utilisateur (*user-driven adaptation*) et adaptation dirigée par le contexte (*context-driven adaptation*).	Exemple d'application ubiquitaire multi-dispositifs pour envoyer des messages texte via un réseau via trois modes de connexion : Wifi, GSM et un cache, lorsqu'il n'y a plus de connexion.	SOA, UPnP.

Table 4.1 : Tableau récapitulatif des middlewares de composition de services

4.3.2. Analyse comparative des middlewares de composition de services

Dans cette section, nous analysons plus en détail la qualité fournie par les différents modèles proposés par les middlewares étudiés précédemment. La qualité de chaque modèle est estimée en termes de réponses qu'il apporte aux caractéristiques et exigences des environnements ambiants. Pour ce faire, nous avons identifié dix points essentiels à savoir : *Abstraction de l'hétérogénéité (HA- Heterogeneity Abstraction)*, *Expressivité du modèle de connaissances (KME- Knowledge Model Expressiveness)*, *Evaluation de la QoS et du contexte (QCE- QoS and Context Evaluation)*, *Gestion des incertitudes (UM- Uncertainty Management)*, *Découverte dynamique de services (DSD- Dynamic Service Discovery)*, *Classification dynamique de services (DSC- Dynamic Service Classification)*, *Sélection dynamique de services (DSS- Dynamic Service Selection)*, *Composition automatique et dynamique de services (ADSC- Automatic and Dynamic Service Composition)*, *Auto-adaptation (SA- Self Adapting)*, et *Gestion spontanés des événements (SEH- Spontanious Events Handling)*. Ces points constituent ainsi la référence pour pouvoir distinguer entre les différents middlewares de composition de services en termes d'adéquation pour les environnements ambiants.

COCOA-PERSE : le contexte d'utilisation (localisation de l'utilisateur, ses activités, l'état de l'environnement, etc.) n'est pas pris en compte par le modèle de connaissances proposé. L'approche proposée semble orientée beaucoup plus vers la qualité de service que vers contexte d'utilisation. Par conséquent, la méthode d'évaluation adoptée s'est limitée aux attributs de la *QoS*. Par ailleurs, les auteurs affirment que l'application *e-movie*, utilisée dans leur scénario d'évaluation, détecte les changements de contexte de l'utilisateur, mais il n'est pas expliqué comment ces changements sont détectés. En outre, l'approche proposée ne traite pas les aspects dynamiques et stochastiques de l'environnement ubiquitaire. Dans COCOA, l'approche de composition de services proposée consiste à intégrer les conversations des services afin de répondre à la conversation de la tâche spécifiée. Lors de ce processus d'intégration, les fonctionnalités requises par une tâche sont liées aux fonctionnalités annoncées par les différents services disponibles. Toutefois, cette approche de composition n'est pas tout à fait automatique puisque la conversation de la tâche (composition finale à atteindre par l'intégration des services) est supposée être connue et établie à l'avance. En outre, le concept événement n'est pas pris en compte par cette approche. Enfin, la méthode d'évaluation est limitée à un scénario simple qui n'intègre pas une large gamme de dispositifs ubiquitaires tels que les capteurs, les actionneurs, etc.

PEIS-Ecology : ce middleware ne propose pas un modèle clair pour évaluer la qualité de service et le contexte d'un composant PEIS ou d'une configuration d'une écologie PEIS. Ainsi, il n'est pas tout à fait claire, ou devrait être étudiée, comment une configuration PEIS générée peut tenir compte des besoins de l'utilisateur, ses préférences et son contexte en général. En outre, les configurateurs PEIS (*PEIS-configurators*) ne gèrent pas l'aspect sélection de services afin de choisir la meilleure configuration PEIS, mais acceptent et instancient simplement la première configuration correcte qu'ils peuvent trouver. Il n'y a pas une estimation à priori des éventuelles faiblesses à long terme de la configuration instanciée. Enfin, le mécanisme d'auto-adaptation proposé est très basique et se limite uniquement au cas où un composant PEIS signale qu'une fonctionnalité utilisée dans la configuration actuelle

n'est plus disponible. Dans ce cas, le composant configurateur (*configurator*) tente de générer une configuration alternative. Cette solution simple doit être étendue pour couvrir d'autres cas plus intéressants notamment lorsqu'un composant PEIS échoue silencieusement, ou lorsque le coût d'une fonctionnalité change dans le sens de rendre une autre configuration préférable. En général, une partie importante du problème d'adaptation à court terme est de savoir comment déterminer le moment où la configuration actuelle doit être changée.

URC-SURF : il y a une limitation dans le modèle de connaissances proposé. Les services sont décrits uniquement dans le modèle OWL-S et il n'y a pas d'extension afin de prendre en compte les concepts de qualité de service, de contexte d'utilisation et d'événements. Ainsi, aucun mécanisme de sélection de services n'est proposé par le middleware URC-SURF. En outre, ce middleware ne tient pas compte de la gestion des événements et ne propose pas une nouvelle approche pour la composition de services, mais il utilise simplement le planificateur SHOP2 existant. Enfin, dans le scénario proposé, l'approche semble être réactive au changement du contexte (localisation de l'utilisateur), mais il n'est pas claire comment ce changement est détecté et quel est le mécanisme d'adaptation qui gère cette réactivité.

MySIM : le modèle de connaissances proposé par ce middleware décrit les propriétés non-fonctionnelles de la qualité de service et leurs méthodes d'évaluation. Toutefois, ce modèle ne spécifie pas le contexte d'utilisation d'une façon générale et ne tient pas compte de la notion d'événements. En outre, le mécanisme d'adaptation proposé est déclenché uniquement en cas d'apparition d'un service (déploiement d'un nouveau service) et/ou de disparition d'un service (service désinstallé ou arrêté). Ce mécanisme nécessite de tenir compte de plusieurs autres événements imprévisibles qui peuvent se produire dans un environnement ubiquitaire tels que le service en cours d'utilisation tombe en panne ou sa qualité diminue, changement des besoins de l'utilisateur ou de son contexte, etc. Enfin, l'approche de communication adoptée se base sur la plate-forme OSGi. Cependant, les auteurs ne détaillent pas l'implémentation du middleware MySIM à base d'OSGi afin de supporter la communication entre les dispositifs ubiquitaires.

MEDUSA : les auteurs supposent ici que chaque environnement ubiquitaire fournit un ensemble de cartes associées aux instances des services disponibles dans l'environnement. Cette hypothèse semble faisable dans des petits espaces avec un nombre limité de services, mais elle reste difficilement applicable dans un environnement ubiquitaire avec un nombre élevé de services qui peuvent apparaître et disparaître dynamiquement. En outre, le plan de composition de service (application) est conçu manuellement de manière abstraite en utilisant un outil dédié à la composition. Les services constituant l'application conçue sont connectés au moment de l'exécution aux instances des services disponibles dans l'environnement par le compositeur d'application et contrôlés par les utilisateurs grâce à un ensemble d'interfaces utilisateur. Cependant, cette approche de composition ne tient pas compte des changements de contexte, surtout quand une instance d'un service connecté devient indisponible ou tombe en panne au moment de l'exécution. Des mécanismes d'adaptation devraient alors être intégrés dans le moteur de composition pour un monitoring plus approprié des plans de composition de services. Enfin, MEDUSA ne traite pas les événements de l'environnement.

MUSIC : les connaissances sont décrites dans un modèle sensible à la QoS. Bien que ce modèle décrive certains aspects pertinents de la qualité de service, il ne tiens pas compte de la notion d'événements et du contexte utilisateur tels que ses activité, sa localisation, son environnement, etc. Une extension est alors nécessaire pour ce modèle afin d'intégrer ces dimensions contextuelles. Le contexte doit être aussi pris en compte par le modèle de sélection de services qui est guidé par la fonction d'utilité fournie par le développeur. Cette fonction vise à satisfaire au mieux l'utilisateur en tenant compte de ses préférences exprimées comme des poids associés à chaque paramètre de qualité. En outre, les plans de composition de services sont prédéfinis et décrits de façon abstraite dans le modèle sensible à la QoS. Dans ce modèle, la planification se réfère au processus de sélection du meilleur plan de composition (i.e. configuration de l'application) en termes d'utilité pour l'utilisateur final. Ainsi, l'approche de composition proposée n'est pas entièrement automatisée et requiert comme entrées des plans de composition de service préétablis. Enfin, lorsque le processus d'adaptation est déclenché par un changement de contexte, l'espace de recherche pour un plan alternatif se limite uniquement aux plans préétablis initialement. Ainsi une telle adaptation risque d'échouer s'il n'y a aucun plan qui peut satisfaire le nouveau contexte détecté.

SeSCO : dans le modèle de connaissances proposé, le contexte d'utilisation n'est pas intégré dans la représentation des services et des tâches. En outre, l'approche proposée ne tient pas compte de la notion d'événement que ce soit en termes de modélisation ou de gestion. De plus, malgré que l'approche d'adaptation proposée tienne compte de l'arrivée et de départ dynamique des dispositifs, les auteurs n'expliquent pas comment cette approche s'auto-adapte aux changements du contexte notamment au cas de panne d'un plan de composition de services. Enfin, il n'y a pas une proposition de scénario afin de démontrer l'intérêt et la faisabilité de l'approche proposée dans un environnement ubiquitaire réel.

SeGSeC : dans le modèle de connaissances proposé, les composants sont représentés à l'aide des graphes sémantiques qui comportent des nœuds et des liens marqués. Cependant, cette représentation se focalise beaucoup plus sur la définition sémantique des composants en termes de leurs opérations et leurs paramètres d'entrées/sorties sans tenir compte des attributs de la *QoS* et du contexte. Ainsi, la méthode de sélection proposée se base uniquement sur les paramètres fonctionnels des composants et n'intègre ni la qualité de service, ni le contexte d'utilisation. De plus, la requête de l'utilisateur est initialement spécifiée dans un langage naturel. Par la suite, elle est transformée en une représentation sous forme d'un graphe sémantique CoSMoS en utilisant des techniques existantes. Cependant, les auteurs ne précisent pas le type d'information que l'utilisateur doit spécifier dans sa requête en langage naturel. L'approche de composition de service proposée crée un *workflow* en se basant sur la compatibilité des prédicats. Néanmoins, il n'y a pas de détails sur la méthode de *matching* utilisée pour vérifier cette compatibilité entre prédicats. Après sa création, le *workflow* sera examiné si sa sémantique répond à la requête de l'utilisateur. Dans le cas ou la sémantique de ce *workflow* ne satisfait pas la requête de l'utilisateur, le mécanisme d'adaptation tente de créer d'autres *workflows* en se basant sur la même requête. Toutefois, ce mécanisme d'adaptation ne répond pas aux exigences ubiquitaires, notamment lorsqu'un ou plusieurs composants du *workflow* tombent en panne lors de l'exécution. Enfin, les dispositifs

impliqués dans l'exemple de scénario proposé pour valider l'approche sont limités à des PCs et PDAs.

URS : le modèle de connaissances proposé manque de descriptions formelles des attributs de la *QoS*. Par conséquent, ce middleware ne propose aucune méthode d'évaluation de la QoS et de sélection de services. En effet, lorsque plusieurs services composites sont construits sous formes des CINs (réseau d'interprétation du contexte) pour décrire une même situation courante, il n'est pas clair comment choisir le service composite qui interprète au mieux cette situation. En outre, l'approche proposée ne tient pas compte du caractère incertain et stochastique de l'environnement. Enfin, URS ne propose pas un mécanisme de gestion spontanée des événements qui peuvent se produire d'une manière imprévisible dans l'environnement ambiant.

VRESCo : dans le modèle de connaissances proposé, un service est défini par ses caractéristiques fonctionnelles et non-fonctionnelles. Des formules de calcul des attributs non-fonctionnels sont fournies pour les cas déterministes et non-déterministes. Toutefois, ce modèle ne tient pas compte des spécifications du contexte d'utilisation. En outre, le problème de la sélection de services est formulé comme un problème de satisfaction de contraintes (CSP), qui prend comme entrées tous les services candidats regroupés par fonction. Des poids sont associés à chaque niveau hiérarchique des contraintes afin de trouver une solution optimale au CSP formulé. Néanmoins, la structure initiale du service composite est spécifiée manuellement par l'utilisateur. Enfin, un simple scénario de composition d'un service de guide restaurant est illustré. Ce service composite est utilisé dans les appareils mobiles afin de localiser rapidement et de réserver un restaurant pour un certain nombre de personnes. Cependant, les auteurs n'intègrent pas assez de dispositifs ubiquitaires dans ce scenario et ne proposent pas aussi une évaluation des performances de l'approche proposée.

WComp : dans l'approche d'adaptation dirigée par l'utilisateur (*user-driven adaptation*) adoptée par WComp, un utilisateur peut sélectionner un AA (Aspects d'Assemblage) afin d'incorporer ou d'effacer certains comportements et fonctionnalités du système. Deux types d'utilisateurs sont ainsi distingués: les experts et les utilisateurs finaux. Les experts peuvent concevoir de nouveaux AAs pour les nouvelles situations, tandis que les utilisateurs finaux n'ont pas besoin de créer de nouveaux AAs, mais peuvent seulement sélectionner les AAs prédéfinis. Toutefois, cette approche de sélection ne spécifie pas les exigences de l'utilisateur et ne tient pas compte des paramètres de la qualité de service. De plus, dans l'approche de composition utilisant SLCA, un service composite est construit manuellement par assemblage de composants légers. Enfin, le mécanisme d'adaptation à l'aide des AAs fournit une adaptation structurelle au service composite. En d'autres termes, l'assemblage des composants internes du service composite est modifié sans modification de ses composants de base. Donc, cette adaptation peut échouer si certains composants de base tombent en panne lors de l'exécution du service composite.

La **table 4.2** donne une comparaison entre les différents middlewares étudiés par rapport aux défis de la composition de services dans les environnements ambiants.

Frameworks	*HA*	*KME*	*QCE*	*UM*	*DSD*	*DSC*	*DSS*	*ADSC*	*SA*	*SEH*
COCOA	+	+/-	+/-	-	+/-	-	+	+/-	-	-
PEIS	+	+/-	-	+/-	+/-	-	-	+	+/-	+/-
URC	+	+/-	-	+/-	+	-	-	+/-	-	+/-
MySIM	+/-	+/-	+/-	+/-	-	-	+	+	+/-	-
MEDUSA	+	+/-	+/-	-	+	-	+/-	+/-	-	-
MUSIC	+	+/-	+/-	+/-	+	-	+	+/-	+/-	-
SeSCO	+	+/-	-	+	+	-	-	+	+/-	-
SeGSeC	+	+/-	-	-	+/-	-	+/-	+	+/-	-
URS	+	+/-	-	+/-	+	-	-	+	+/-	-
VRESCo	+	+/-	+/-	-	+/-	-	+/-	+/-	-	-
Wcomp	+	+/-	-	+/-	+	-	-	+/-	+/-	+/-

+: OUI (*addressed*); -: NON (*not addressed*); +/-: Pas explicitement (*addressed partially*).

Table 4.2 : Comparaison des middlewares de composition de services

4.4. Conclusion

Dans ce chapitre, nous avons étudié et analysé les middlewares les plus pertinents pour la composition de services dans le domaine de l'intelligence ambiante qui inclue notamment les domaines de l'informatique ubiquitaire et *pervasive* et le domaine de la robotique ubiquitaire. Nous avons décrit l'architecture et le fonctionnement de ces middlewares ainsi que leurs caractéristiques en relation avec la composition de services. Nous avons aussi dressé une comparaison entre ces middlewares par rapport aux défis majeurs de la composition de service dans les environnements ambiants. Nous estimons que, si certains défis et exigences sont relativement bien traités par certains middlewares proposés, d'autres restent encore à un stade préliminaire et doivent être bien étudiés. Premièrement, peu de middlewares supportent la spécification de la qualité de service et du contexte d'utilisation. L'intégration à la fois du contexte et de la *QoS* dans la composition de services permet de fournir le service nécessaire avec la qualité requise en fonction du contexte d'utilisation. Deuxièmement, la gestion des incertitudes est également moins abordée par les middlewares étudiés. L'incertitude est le résultat de la nature dynamique et stochastique des environnements ambiants, elle doit être prise en compte à plusieurs niveaux à commencer du dispositif physique jusqu'au service logiciel. Troisièmement, la plupart des approches proposées pour la composition de services sont manuelles et supposent que le service composite est déjà construit sous forme d'un *Workflow* ou d'un plan de services. Toutefois, avec la prolifération du nombre de services disponibles dans l'environnement, il devient de plus en plus difficile pour les utilisateurs d'analyser et de spécifier manuellement le service composite. Ainsi, les approches automatiques pourraient se substituer à l'analyse humaine en générant automatiquement le service composite. Quatrièmement, la détection et la gestion spontanée des événements ne sont pas prises en compte par la plupart des middlewares proposés, et ce, malgré l'importance de cet aspect dans la gestion des environnements ambiants. Cinquièmement, la sélection et la découverte dynamique de services demeurent toujours des problèmes ouverts et ne sont pas bien intégrées par la majorité des middlewares présentés dans ce chapitre. En effet, la découverte dynamique des services et des ressources permet de répondre aux besoins d'un environnement volatile, où les services peuvent apparaître et disparaître dynamiquement

tandis que la sélection dynamique de services permet de répondre en permanence aux besoins des utilisateurs et d'améliorer la qualité de service fournie et ainsi augmenter le degré d'acceptabilité du service composite. Finalement, les mécanismes prévus pour l'auto-adaptation des services composites présentent beaucoup de limites. En effet, ces mécanismes sont moins adaptables aux changements continuels du contexte d'utilisation et aux perturbations qui peuvent survenir sur les services, notamment l'apparition et la disparition de services lors de l'exécution.

Compte tenu de cette étude et analyse des points forts et points faibles des middlewares de composition de services existants, nous proposons une nouvelle vision qui tient compte, au mieux, des différentes lacunes et faiblesses de ces middlewares. Notre vision du middleware de composition de services dans un environnement ambiant fera ainsi l'objet du prochain chapitre de ce manuscrit.

PARTIE II : PROPOSITIONS

CHAPITRE 5

Modèle des connaissances et un Framework orienté services

5.1. Introduction

Dans la première partie portant sur l'état de l'art, nous avons étudié et analysé les différents middlewares de composition de services dans un environnement ambiant selon les dix modèles orientés services cités dans la **section 2.6.2 du chapitre 2** à savoir : *architecture générale, modèle des connaissances, modèle d'évaluation de la QoS et du contexte, modèle de matching des services, modèle de sélection des services, modèle de découverte des services, modèle de composition des services*, *modèle de monitoring des services*, *modèle d'évaluation*, et *modèle de communication*. Suite à cette étude, nous avons décelé d'énormes lacunes quant à la prise en compte effective des caractéristiques et des exigences des environnements ambiants qui soulèvent des défis considérables pour la composition de services. Dans cette présente partie qui porte sur notre proposition, nous décrivons en détail notre vision et conception du système de composition de services dans un environnement ambiant. Cette proposition tient compte des différentes lacunes identifiées et en apporte les réponses nécessaires afin de réaliser concrètement les dix modèles orientés services cités précédemment et les intégrer dans une architecture cohérente. Pour ce faire, la partie proposition est structurée autour de trois chapitres selon l'ordre suivant : le premier chapitre porte sur le *modèle des connaissances* et l'*architecture générale ;* le deuxième chapitre décrit la réalisation technique des modèles suivants : *le modèle de découverte des services, le modèle de composition des services, le modèle de sélection des services, le modèle d'évaluation de la QoS et du contexte, le modèle de matching des services, et le modèle de monitoring des services* ; et enfin le troisième et dernier chapitre de validation porte sur le *modèle d'évaluation* et le *modèle de communication*.

Dans ce premier chapitre, nous décrivons le modèle des connaissances et l'architecture de notre système de composition de services. En premier lieu, le modèle des connaissances constitue le support sur lequel repose tous les autres modèles. Son objectif est de modéliser toute la connaissance dans des formats intelligibles et exploitables par les autres modèles. La connaissance elle même est scindée en trois parties : les données, les services et les tâches. Les données sont subdivisées en données fonctionnelles et non-fonctionnelles. Les services, quant à eux, sont scindés en services abstraits et services concrets. Concernant les tâches, le modèle de connaissance distingue les événements et les requêtes. En second lieu, l'architecture générale donne une vision globale sur le rôle, le fonctionnement et l'organisation des différents modules au sein de notre système de composition de services. Ce système est décrit comme un Framework de composition de services sensible aux événements appelé *FrEvASeC* (***Fr**amework for **Ev**ents **A**ware **Se**rvice **C**omposition).*

Ainsi, l'organisation de ce présent chapitre est scindée en deux grandes parties. La première partie définie et formalise les différents concepts adoptés par le modèle des connaissances. La deuxième partie décrit, quant à elle, notre vision du système de composition de services dans un environnement ambiant. Cette vision repose sur un Framework comprenant plusieurs modules interconnectés et engagés dans un processus de collaboration afin de réaliser les objectifs spécifiés lors de la détection des événements dans un environnement ambiant. Ainsi, le fonctionnement de ce Framework est détaillé par la description des différentes relations entre les modules dont il se compose.

5.2. Description du modèle des connaissances

5.2.1. Organisation générale des connaissances

Les connaissances manipulées dans un environnement ambiant peuvent être scindées en trois grandes parties : les données, les services et les tâches (**Figure 5.1**). Les données sont subdivisées en données fonctionnelles et non-fonctionnelles. Les données fonctionnelles comprennent les données contextuelles et les messages échangées entre les services, tandis que les données non-fonctionnelles comprennent les spécifications des utilisateurs ainsi que les paramètres de qualité de service (*QoS*) qui peuvent être statiques ou dynamiques. Les services, quant à eux, sont scindés en services abstraits et services concrets. Ces derniers appartiennent à l'une des trois catégories de services suivantes : services sensibles aux événements (*Event-aware service*s), services sensibles au contexte (*Context-aware services*), et services actionneurs (*Devices control services*). La classe des services sensibles au contexte, elle-même, imbrique deux sous classes de services : les services d'acquisition du contexte (*Context acquisition services*) et les services de traitement du contexte (*Context processing services*). Concernant les tâches, le modèle de connaissances distingue entre les tâches déclenchées par les requêtes et les tâches déclenchées par les événements. Les tâches déclenchées par les requêtes sont des tâches explicitement spécifiées par les utilisateurs au système en online, tandis que les tâches déclenchées par les événements sont des tâches implicites et préalablement intégrées au système afin de réagir à la détection des événements.

5.2.2. Représentation des données

5.2.2.1. Les données fonctionnelles

Les données fonctionnelles sont des données qui alimentent l'ensemble des services de l'environnement ambiant afin qu'ils puissent avoir un fonctionnement correct. Nous considérons que toute donnée fonctionnelle de l'environnement ambiant véhicule un certain contexte potentiel soit sur l'environnement, sur les utilisateurs ou bien sur les dispositifs. Une donnée fonctionnelle est alors vue comme un paramètre de contexte tandis qu'un ensemble de données fonctionnelles structurées est vue comme un message contenant du contexte.

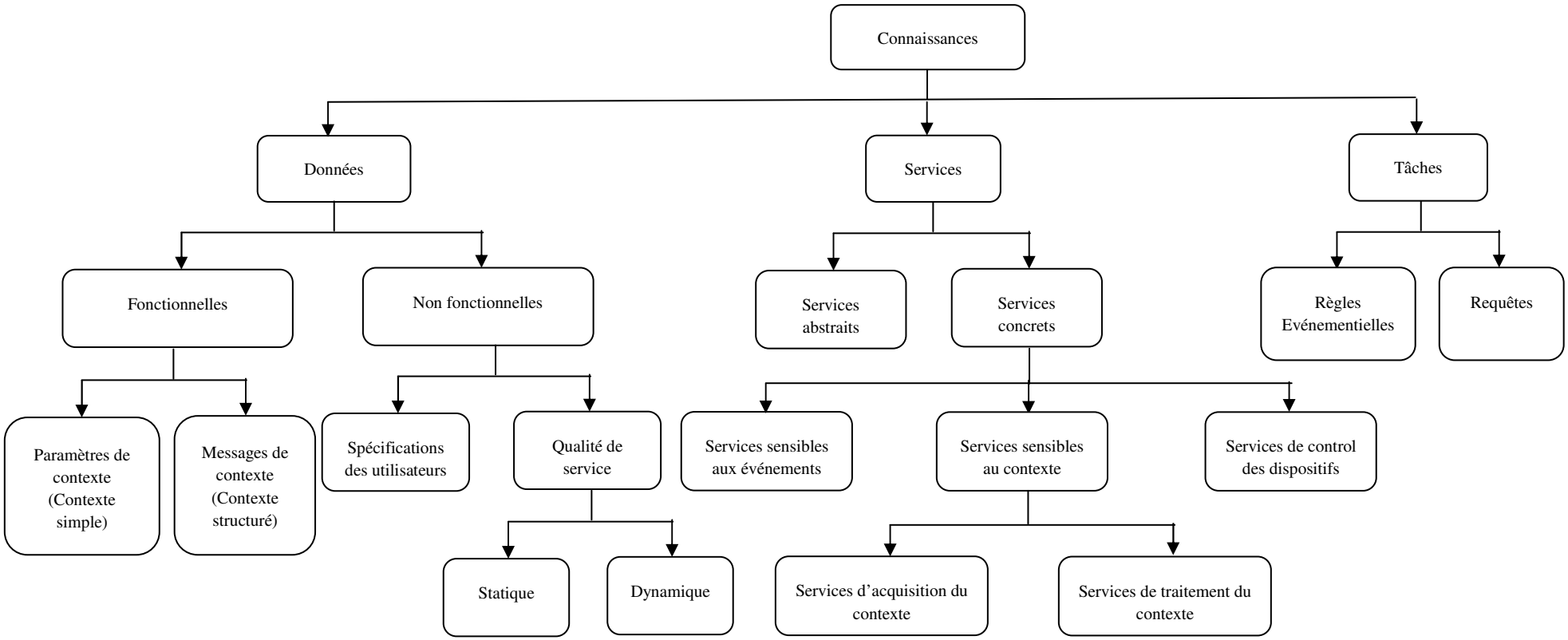


Figure 5.1 : Organisation générale de la connaissance dans un environnement ambiant

Les paramètres de contexte : Un paramètre de contexte est une information contextuelle brute ou traitée qui est acquise de l'environnement ambiant. Une information contextuelle est un concept *Cpt* qui est représenté par un triplet <*nom, type, sémantique*> tels que : *nom* représente le nom syntaxique de *Cpt*, *type* représente le type syntaxique de *Cpt*, et *sémantique* représente la description sémantique de *Cpt* dans une ontologie, i.e. une hiérarchie de concepts. *Sémantique* lui même est représenté par un quadruplet <*O, Position, Pères, Fils*> avec: *O* est l'ontologie de concepts à laquelle *Cpt* appartient ; *Position* est l'emplacement (*profondeur*, *largeur*) du concept *Cpt* dans l'ontologie *O* tels que *: profondeur* est le niveau du concept *Cpt* en explorant l'ontologie *O* en profondeur, et *largeur* est le niveau du concept *Cpt* en explorant l'ontologie *O* en largeur de gauche à droite au niveau spécifié par *profondeur* ; *Pères* est l'ensemble des concepts pères de *Cpt* et *Fils* l'ensemble des concepts *Fils* de *Cpt*. Pour le calcule de la *Position*, on admet que la *profondeur* initiale (le premier niveau) d'une ontologie est égale à 0 et la *largeur* se calcule de gauche à droite en commençant toujours de 0 pour chaque *profondeur* de l'ontologie. Ainsi, les *Pères* d'un concept *Cpt* dans une ontologie *O* est l'ensemble des concepts qui sont liés directement ou indirectement au concept *Cpt* dans l'ontologie *O* et qui appartiennent à des profondeurs inférieures à celle de *Cpt*. Les *Fils* d'un concept *Cpt* dans une ontologie *O* est l'ensemble des concepts qui sont liés directement ou indirectement au concept *Cpt* dans l'ontologie *O* et qui appartiennent à des profondeurs supérieurs à celle de *Cpt*. Une relation est directe entre deux concepts dans une ontologie si ces deux concepts sont liés par une relation de type « est-un » (*is-a*). Une relation est indirecte entre deux concepts dans une ontologie si au moins deux relations successives de type « est-un » s'intercalent entre ces deux concepts dans l'ontologie. La **figure** 5.**2** illustre un exemple d'une ontologie de concepts relatifs au paramètre de contexte *adresse*. Cette ontologie est désignée par le nom « *O1* ». Ainsi, le concept *adresse* représente la racine de l'ontologie *O1*et il se situe à la position *(0,0)*. Le concept *adresse logique* est un *fils direct* du concept *adresse* tandis que les concepts *adresse mail* et *adresse IP* sont des *fils indirects* du concept *adresse* et des *fils directs* du concept *adresse logique*. Prenant par exemple le concept *adresse mail*. Ce concept est représenté par le triplet suivant < *adresse mail, string, sémantique*> avec : *sémantique* est représenté par le quadruplet suivant : <*O1, (2,0), {adresse logique, adresse}, ∅*>.

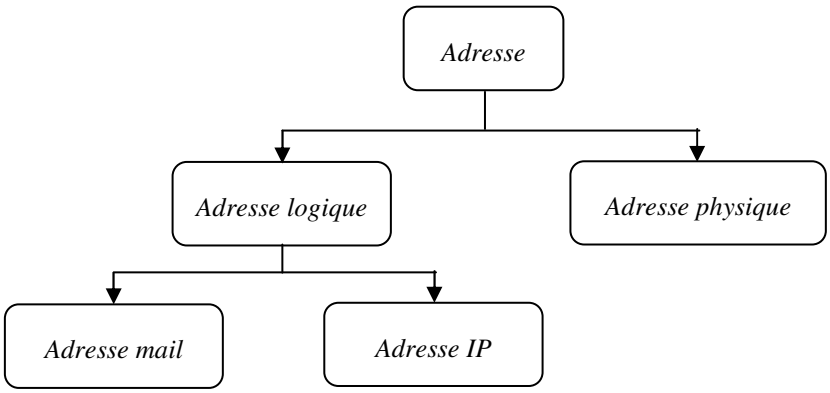


Figure 5.2 : Exemple d'une ontologie du concept « adresse »

Un concept dans une ontologie peut être soit un *concept terminal* soit un *concept intermédiaire*. Un *concept terminal* est un concept appartenant à une profondeur terminale

(dernier niveau) d'une branche dans l'arborescence décrivant l'ontologie. Un *concept terminal* est aussi appelé *feuille* de l'ontologie car il représente le dernier concept d'une branche qui parte de la racine de l'ontologie. Un *concept intermédiaire* est, quant à lui, un concept qui ne se situe pas sur les feuilles de l'arborescence décrivant l'ontologie. Prenant l'exemple de l'ontologie décrite sur la **figure 5.2**, les concepts *adresse* et *adresse logique* sont des *concepts intermédiaires* tandis que les concepts *adresse physique*, *adresse mail* et *adresse IP* sont des *concepts terminaux.*

Dans une ontologie, il n'ya que les *concepts terminaux* (feuilles) qui possèdent un type syntaxique. En effet, ces concepts représentent l'instanciation concrète de tous les concepts de l'ontologie. Lorsqu'un concept est *intermédiaire* (i.e. n'est pas une feuille) dans l'arborescence décrivant une ontologie, nous convenons de spécifier son type comme *abstrait*. En effet, un *concept intermédiaire* existe uniquement au niveau sémantique et ne possède pas un type syntaxique. Dans la réalité, un *concept intermédiaire* ne peut pas exister au tant que tel, mais son existence est incarnée par ses *fils directs* si ces derniers sont des feuilles sinon il est incarné par ses *fils indirects* qui sont des feuilles. Ces derniers sont obtenus par instanciation successives de tous les *concepts fils* qui sont *intermédiaires* jusqu'à obtenir des concepts *fils* qui sont des *terminaux*. Reprenant l'exemple de l'ontologie décrite sur la **figure 5.2**, une *adresse physique* et une *adresse mail* peuvent être de type *string* et une *adresse IP* peut être de type *entier* avec un format spécifique. Par contre, les concepts *adresse* et *adresse logique* n'ont pas un seul type particuliers, donc leur type reste *abstrait*. En effet, le concept *adresse* peut avoir trois instanciations concrètes différentes (fils directs et indirects): *adresse physique, adresse mail ou adresse IP.* Tandis que le concept *adresse logique* peut avoir uniquement deux instanciations concrètes différentes : *adresse mail* ou *adresse IP.* Ainsi, les types syntaxiques des concepts *adresse* et *adresse logique* dépendront des instanciations concrètes de ces concepts. Par exemple, le type syntaxique du concept *adresse logique* est soit *string* si le concept *adresse logique* est instancié comme *adresse mail* ou bien *entier (format spécifique)* si le concept *adresse logique* est instancié comme *adresse IP.*

La structuration des paramètres de contexte dans des ontologies claires et compréhensibles constitue la première étape pour une analyse sémantique du contexte. Plusieurs ontologies peuvent êtres élaborées selon les paramètres de contexte à traiter. La **figure** 5.**3** représente un autre exemple d'une ontologie des identifiants utilisés dans un environnement ambiant. Dans un tel environnement, un identifiant (ID) est associé à chaque objet et personne afin de pouvoir les identifier de manière unique. La valeur de l'identifiant est parfois portée par l'objet lui-même. Cela est possible notamment avec les objets de type robots, capteurs ou caméras. Car ces derniers possèdent la capacité de sauvegarder et de communiquer leur identifiants. Par ailleurs, lorsqu'il s'agit d'une personne ou d'un objet n'ayant pas cette capacité de sauvegarder un identifiant, un petit objet de type *Tag RFID* ou de type *capteur* leur seront associés afin de les identifier. Toutefois, la structure de l'identifiant dépend de l'objet qui le sauvegarde et le communique. Par exemple, un identifiant d'un capteur de type *Imote2* est déférent de celui d'un capteur de type *TelosB* et de celui des capteurs de type *Cricket.* De même un identifiant d'un *Tag actif* est différent de celui d'un *Tag passif*. Par ailleurs, si on admet que les autres objets tels que les robots et les PC ont

des identifiants générique qui ont la même structure, alors on aura l'arborescence des identifiants spécifiée par l'ontologie de la **figure 5.3**.

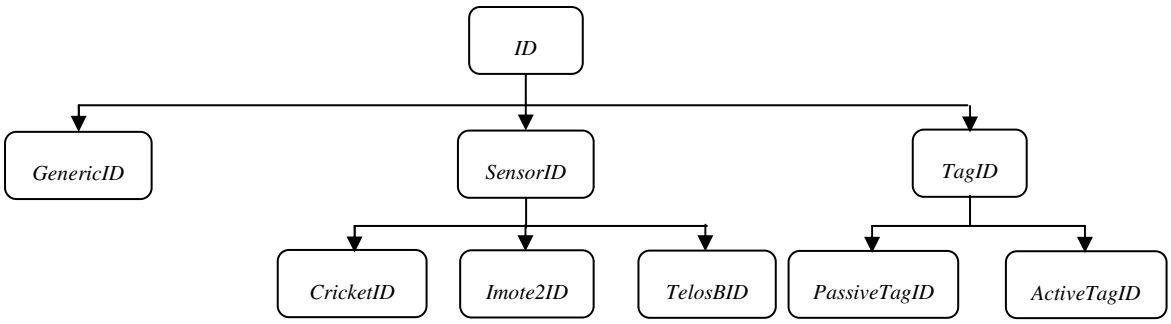


Figure 5.3 : Exemple d'une ontologie du concept « ID »

Par ailleurs, il est tout à fait possible d'avoir une ontologie de toutes les entités qui existent dans un environnement ambiant. Nous pouvons catégoriser ces entités en entités physiques et entités non physiques. Dans les entités physiques, on distingue entre les personnes et les autres objets. Ces derniers peuvent être soit des objets mobiles ou bien des objets statiques (fixes). Les entités non physiques concernent tous les softwares disponibles soit sous forme de services ou d'applications. Ainsi, l'ontologie de la **figure 5.4** illustre une catégorisation des différentes entités existantes dans un environnement ambiant.

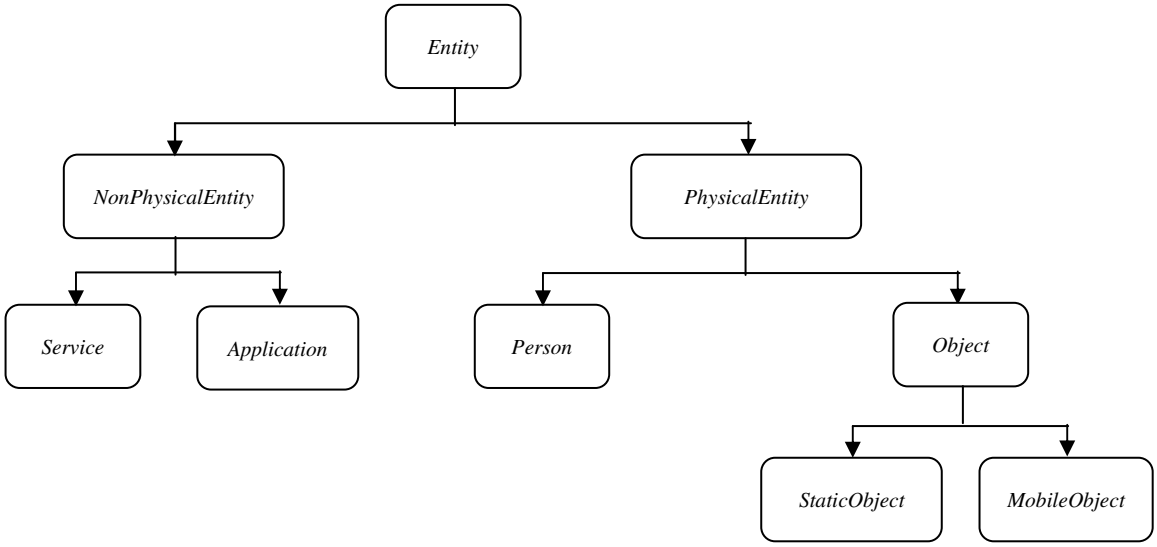


Figure 5.4 : Exemple d'une ontologie du concept « Entité »

Afin de gérer une seule ontologie globale, nous définissons une racine conventionnelle commune appelée *AmbientEnvironment*, à laquelle sont liées toutes les racines des différentes ontologies d'un même domaine par des relations de type « est-un ». Par exemple, pour unifier les trois ontologies décrites précédemment à savoir : l'ontologie des adresses (**Figure 5.2**), l'ontologie des identifiants (**Figure 5.3**) et l'ontologie des entités (**Figure 5.4**), il suffit de lier les trois concepts suivants : *Adresse, ID* et *Entity* au concept conventionnel *AmbientEnvironment* par trois relations de type « est-un ».

Les messages de contexte : Un message est une structure de données qui encapsule des paramètres de contexte. Dans un environnement ambiant, un message véhicule un certain

contexte satisfait à un instant donné. Ainsi, un message *M* est représenté par un couple : < *timestamp, paramètres de contexte*> tels que : *timestamp* représente l'instant de délivrance du message *M* par sa source et *paramètres de contexte* représente une liste de toutes les données fournies par le message *M*. Chaque donnée est un concept appartenant à une certaine ontologie et sa représentation respecte la spécification des paramètres de contexte décrite dans la section précédente. Un ensemble de paramètres de contexte sont encapsulés dans un même message si ces paramètres présentent des dépendances mutuelles quant à leur utilisation. Un paramètre de contexte *Cpt1* dépend d'un autre paramètre de contexte *Cpt2* si l'utilisation et l'exploitation de *Cpt1* nécessitent la connaissance de *Cpt2*. Autrement dit, l'information véhiculée par *Cpt1* reste incomplète sans la connaissance de *Cpt2*. Soit par exemple, un paramètre de contexte *distance* qui est encapsulée dans le message suivant : <*timestamp, distance*>. Ce message nous indique une distance mais on ignore les deux points par rapport auxquels cette distance est mesurée. Donc, ce message reste incomplet et inexploitable. Ainsi, le message construit sera de la forme <*timestamp, ID1, ID2, distance*> avec *ID1* et *ID2* représentent les identifiants des deux points (objets) de mesure. Afin de séparer clairement entre les messages et faciliter leur utilisation et exploitation, il est nécessaire de construire autant de messages qu'il ya des données indépendantes. Par exemple, le message suivant <*timestamp, température, humidité*> véhicule deux informations contextuelles complètement différentes: l'une sur la température et l'autre sur l'humidité. En réalité, ces deux informations peuvent êtres utilisées l'une indépendamment de l'autre. Si on a besoin de connaitre juste la température, il n'est pas alors nécessaire d'avoir l'humidité et vice-versa. Mais comme elles sont imbriquées dans un même message, ces deux informations sont envoyées à la fois. Donc, l'idée consiste à séparer la température et l'humidité dans les deux messages suivants : <*timestamp, température*> et <*timestamp, humidité*>. De cette façon, il sera plus facile d'exploiter ces deux messages selon le besoin.

Etant donné qu'un message enveloppe un ensemble de concepts qui peuvent êtres des *concepts intermédiaires* ou bien *des concepts terminaux*, alors un message est dit *intermédiaire* si et seulement s'il contient au moins un *concept intermédiaire* et il est dit *terminal* si et seulement si tous les concepts qu'il contient sont tous des *terminaux*. Soient par exemple les deux messages suivants : *M1* < *adresse, adresse physique* > et *M2* < *adresse mail, adresse IP* >. Le message *M1* est un message *intermédiaire* car le concept *adresse* est un *concept intermédiaire* tandis que le message *M2* est un message *terminal* car les deux concepts *adresse mail* et *adresse IP* sont tous les deux des *terminaux*.

5.2.2.2. Les données non-fonctionnelles

Les données non-fonctionnelles sont des données qui ne sont pas nécessaires pour avoir un fonctionnement normal d'un service. Autrement dit, il s'agit de données facultatives dont un service peut s'en passer lors de son exécution. Toutefois, la prise en compte de ces données améliore considérablement la qualité fournie par un système et augmente son degré d'acceptation par les utilisateurs. Dans ce présent travail, nous distinguons deux types de données non-fonctionnelles : les données relatives à la qualité de service et les données relatives aux spécifications des utilisateurs.

Les paramètres de la qualité de service : Les paramètres de la qualité de service *(QoS)* concernent les paramètres de qualité qui sont spécifiés dans la description des services. Chaque paramètre nous renseigne sur une valeur d'une qualité bien précise fournie par un service tels que : la sécurité, le cout, le temps de réponse, etc. Les paramètres de qualité de service peuvent être ajoutés soit comme une sous partie dans la description WSDL d'un service ou bien dans un fichier séparé qui sera publié conjointement avec la description WSDL d'un service. Ce fichier sert à améliorer et enrichir la description WSDL d'un service et peut être référencé à l'intérieur de cette même description WSDL. En fonction de la fréquence de leur changement et de mise à jour, nous distinguons deux types de paramètres de qualité de service: les paramètres de qualité statique (*SQP : static quality parameters*) et les paramètres de qualité dynamique (*DQP : dynamic quality parameters*).

- ***Les paramètres de qualité statique (SQP):*** il s'agit des paramètres de qualité qui restent inchangés pendant une longue période de temps. En d'autres termes, la fréquence de mise à jour des valeurs de ces paramètres est très petite. Parmi ces paramètres, on peut citer à titre d'exemple, *le niveau de sécurité d'un service, le coût d'un service*, etc. Ces paramètres sont généralement spécifiés lors du déploiement et de la publication des services par les prestataires de service ou pendant la phase de mise en place du contrat d'utilisation des services.
- ***Les paramètres de qualité dynamique (DQP):*** contrairement aux paramètres de qualité statique, les paramètres de qualité dynamique changent en fonction du contexte d'utilisations, par exemple, la disponibilité d'un équipement dépend de son niveau de batterie, un service de vidéos dépend de la connexion du dispositif caméra, etc. Ce type de paramètres change fréquemment et estime l'incertitude sur la réponse de chaque service. Par exemple, le temps de réponse d'un service peut changer en fonction de plusieurs paramètres tels que, l'état de la connexion réseau, la disponibilité du service, le taux d'occupation du serveur, etc.

Les spécifications des utilisateurs : Les spécifications des utilisateurs concernent les paramètres qui indiquent les besoins et les préférences des utilisateurs en termes de la qualité de services requise par leurs applications. En effet, la qualité de service est étroitement liée à l'application considérée. Par exemple, certaines applications nécessitent un haut niveau de sécurité en raison de la confidentialité des données qu'elles manipulent. D'autres applications nécessitent un temps de réponse très court et borné telles que les applications temps réel. On peut citer également les applications embarquées qui tiennent aussi compte de l'énergie des dispositifs qu'elles utilisent. Cette différence en terme d'importance des paramètres de qualité de service peut être décrite par un ensemble de poids *Wi* associés pour chaque paramètre *i* de qualité de service, et ce, en fonction de l'application souhaitée et du contexte d'utilisation. Concrètement, ces poids peuvent êtres initialement spécifiés par l'utilisateur pour refléter l'importance relative de chaque paramètre de qualité de service par rapport à l'application considérée.

5.2.3. Représentation des services

Dans ce travail, nous considérons deux types de services à savoir : les *services concrets* et les *services abstraits*. Un *service concret* est un service qui réalise concrètement une certaine

fonctionnalité tandis qu'un *service abstrait* représente une classe de *services concrets* qui sont fonctionnellement équivalents, i.e. fournissent les mêmes fonctionnalités.

5.2.3.1. Les services concrets

Nous définissons un *service concret* comme étant une fonction physique ou logique qui fourni des données contextuelles et qui peut avoir un effet sur le monde réel. D'une part, un *service concret* est vu comme une fonction logique si son rôle consiste à effectuer des opérations sur le contexte telles que : la production, la transformation et le traitement du contexte. Par exemple, le service *getTemperature* permet de fournir la valeur du contexte *température*. D'autre part, un *service concret* est vu comme une fonction physique s'il permet de réaliser une opération qui produise un effet sur le monde réel. Par exemple, le service *turnOnLight* permet d'allumer une lumière afin d'augmenter le niveau de luminosité du monde réel. Par ailleurs, un *service concret* possède une implémentation réelle invocable via des messages d'entrée afin de produire des messages de sorties. Cette implémentation peut avoir 0 à n messages d'entrée et 1 à n message de sortie. Ainsi, un *service concret* peut jouer, à la fois, le rôle de producteur de contexte sous forme de messages de sortie et le rôle de consommateur de contexte via des messages d'entrée.

Dans un environnement ambiant, nous distinguons trois catégories de services concrets (**Figure 5.5**): les *services sensibles aux événements* (*Event-aware services*), les *services sensibles au contexte* (*Context-aware services*) et les *services de control des dispositifs, i.e. services actionneurs (Devices control services)*. Dans la catégorie des *services sensibles au contexte*, nous distinguons les *services d'acquisition du contexte* (*Context acquisition services*) et les *services de traitement du contexte* (*Context processing services*). D'abord, les *services sensibles aux événements* détectent les événements qui se produisent dans le monde réel et transmettent des notifications à chaque détection d'un type d'événements. Ensuite, les *services sensibles au contexte* se chargent de l'acquisition et du traitement de tout type de contexte. Enfin, les *services de control* produisent des actions et des effets sur le monde réel par le control et la commande des différents dispositifs de l'environnement ambiant. Dans ce travail, les *services sensibles aux événements* et les *services de control de dispositifs* seront utilisés respectivement comme étant des déclencheurs et des actionneurs dans un système intelligent ambiant. Tandis que les *services sensibles au contexte* seront utilisés dans le processus de composition de services sensibles au contexte.

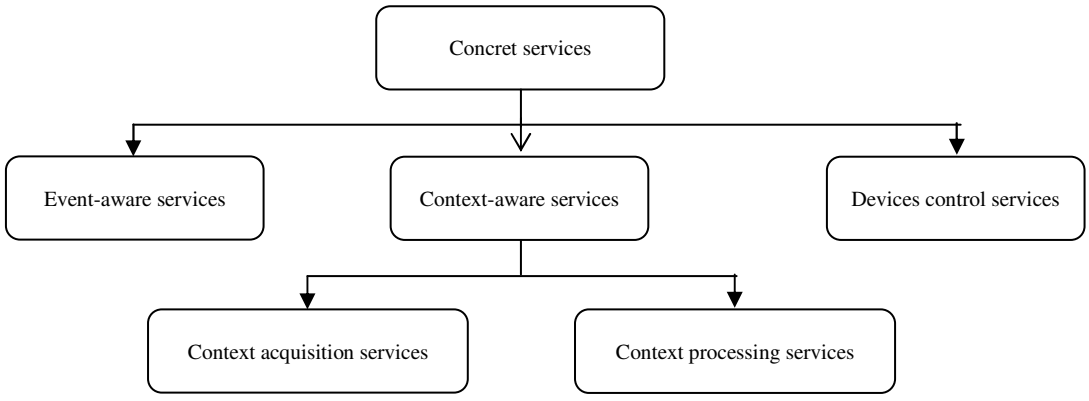
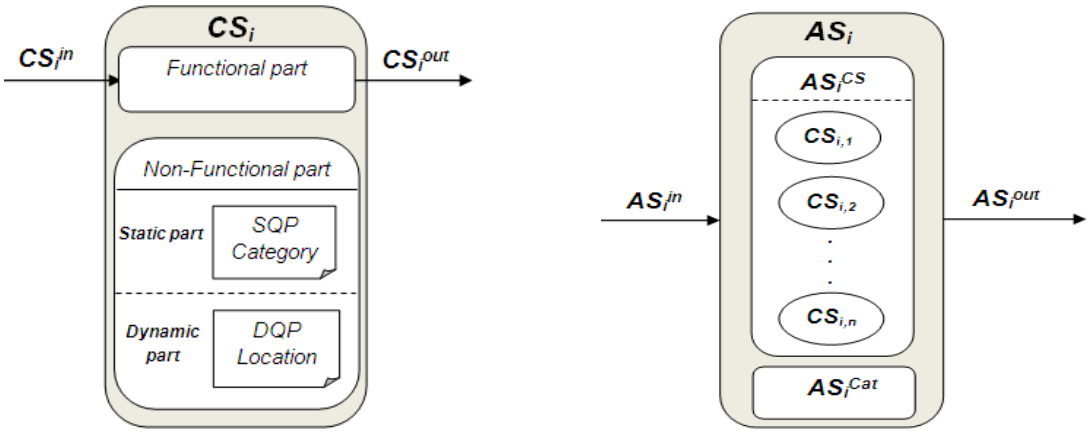


Figure 5.5: Catégories des services concrets

D'un point de vu formel, un service concret *i,* noté cs_i, est subdivisé en deux parties : la partie fonctionnelle et la partie non-fonctionnelle comme le montre la **figure 5.6.a**. La partie fonctionnelle est alimentée par des messages d'entrée (cs_i^{in}) et produit des messages de sortie (cs_i^{out}). La partie non-fonctionnelle est, quant à elle, subdivisée en deux sous parties : la partie statique et la partie dynamique. La partie statique contient les paramètres de qualité de service statique (*SQP*) et indique la *catégorie* du service tandis que la partie dynamique contient les paramètres de qualité de service dynamique (*DQP*) et indique la *localisation* du service. La *localisation* d'un service représente, en quelque sorte, le périmètre ou le champ d'action de ce service. Dans un environnement ambiant, le champ d'action d'un service représente la zone dans laquelle ce service est déployé. Par exemple, dans une maison intelligente, le champ d'action d'un service caméra représente tout le champ de vision de la caméra (couloir, chambre, etc.). Ainsi, la *localisation* d'un service est dynamique car elle change en fonction de la zone du déploiement du service.

Un service concret peut être invoqué (exécuté) si et seulement si tous ses messages d'entrée sont disponibles et sa localisation correspond à l'emplacement dans lequel on désir exécuté l'action (zone d'intérêt). En outre, un service concret est invoqué pour utiliser au moins un de ses messages de sortie. Cela signifie que tous les messages d'entrée sont obligatoires pour l'invocation d'un service concret, alors que ces messages de sortie peuvent être utilisés partiellement. Par ailleurs, tous les paramètres de contexte encapsulés dans les messages d'entrée et de sortie d'un service concret doivent apparaitre dans une ontologie globale. Si certains d'entre eux n'apparaissent pas, alors il est nécessaire de les ajouter à l'ontologie globale avant qu'ils soient manipulés par le service concret.

Le développement d'un nouveau service concret peut tenir compte des services déjà existants afin d'augmenter son utilité pour l'environnement. L'utilité d'un service se mesure en termes de la valeur ajoutée qu'apporte ce service pour l'environnement. Ainsi, l'utilité d'un service concret augmente si, d'une part ce service tend à fournir des messages qui sont très utilisés mais moins produits par les services existants et d'autre part, ce service tend à utiliser des messages qui sont énormément produits mais moins utilisés par les services existants.



(a) (b)

Figure 5.6 : Représentation des services : (a) service concret, (b) service abstrait

5.2.3.2. Les services abstraits

Un *service abstrait* est une classe de services concrets qui sont fonctionnellement équivalents. Par exemple, un service abstrait de température ambiante peut contenir plusieurs services concrets pour mesurer la température dans un environnement ambiant. Chaque service concret est associé à un groupe de capteurs sans fil (ex. un groupe de 4 capteurs). Ainsi, chacun de ces services concrets fournit les différentes valeurs de température échantillonnées par ses propres capteurs dans l'endroit de leur installation (chambre, salon, etc.). L'équivalence fonctionnelle entre les services concrets repose sur trois paramètres à savoir : les messages d'entrée, les messages de sorties et la catégorie des services concrets. Ainsi, deux services concrets sont considérés comme fonctionnellement équivalents (similaires) si et seulement si ces deux services possèdent les mêmes messages d'entrée, les mêmes messages de sortie et appartiennent à une même catégorie de service. Un service abstrait *i* est noté AS_i alors qu'un service concret *j* appartenant au service abstrait *i* est noté $CS_{i,j}$. Comme le montre la **figure 5.6.b**, un service abstrait AS_i est décrit par quatre composantes à savoir: les *messages d'entrée* (AS_i^{in}), les *messages de sortie* (AS_i^{out}), la *catégorie de service* (AS^{cat}) et une *liste de services concrets* (AS_i^{CS}) appartenant au service abstrait AS_i.

Un service abstrait est considéré comme étant une classe de services concrets qui est à la fois abstraite et dynamique. D'une part, un service abstrait est vu comme une classe abstraite car il n'a pas une seule implémentation réelle. En effet, son existence réelle est incarnée par plusieurs implémentations qui correspondent à celles des services concrets qu'il contient. D'autre part, un service abstrait est vu comme une classe dynamique car sa réalisation réelle dépend du service concret choisi lors de la phase d'exécution. Ce choix peut tenir compte de plusieurs paramètres notamment ceux du contexte d'utilisation. Ainsi, l'implémentation d'un service abstrait change dynamiquement en fonction du service concret choisi à un instant donné. Le reste des détailles sur les services abstraits notamment la définition formelle de l'équivalence fonctionnelle entre deux services concrets ainsi que la manière de classifier les services concrets dans des services abstraits feront l'objet de **la section 6.2.2 du chapitre 6**.

5.2.4. Représentation des tâches

Nous considérons une tâche comme étant un objectif à réaliser. Un objectif peut être spécifié sous deux formats différents : un format orienté données et un format orienté services. Le format orienté données décrit les données qu'on souhaite obtenir et éventuellement certaines conditions sur les valeurs de ces données. Le format orienté services décrit, quant à lui, les services qu'on souhaite exécuter et éventuellement certaines conditions sur les valeurs des sorties de ces services. Soit un objectif *G* (*Goal*) qu'on désir spécifier dans les deux formats précédents. Dans le format orienté données, *G* est spécifié par deux ensembles sous le format suivant *G (GM ; GC)* tels que : *GM* (*Goal Messages*) est l'ensemble des messages désirés et *GC* (*Goal Conditions*) est l'ensemble des conditions sur les paramètres contenus dans les messages de *GM*. Soit par exemple, un message *M <suspectObjectAlert>* qui contient le paramètre de contexte *suspectObjectAlert* de type booléen qui indique si un objet suspect est détecté ou non. Si on spécifie l'objectif *G* sous le format orienté données suivant *G ({M} ; {M.suspectObjectAlert = true}*), cela signifie qu'on

souhaite obtenir le message *M* lorsqu'un objet suspect est détecté. Par ailleurs, dans le format orienté services, *G* est également spécifié par deux ensembles sous le format suivant *G (GS ; GC)* tels que : *GS* (*Goal Services*) est l'ensemble des services qu'on désir exécuter et *GC* (*Goal Conditions*) est l'ensemble des conditions sur les paramètres contenus dans les messages de sorties des services appartenant à *GS*. Soit par exemple, un message *M <suspectObjectAlert>* est délivré par le service S comme message de sortie alors *G* sous le format orienté données suivant *G ({S} ; {S.M.suspectObjectAlert = true}*), cela signifie qu'on souhaite exécuter le service S et obtenir le message *M* lorsqu'un objet suspect est détecté.

Dans ce présent travail, on retient le format orienté données pour la spécification des objectifs. En effet, le format orienté données est le format le plus générique car il se focalise sur la donnée plutôt que sur le service à exécuter. En d'autres termes, ce format ne s'intéresse pas à la manière dont la donnée désirée est obtenue. Dans la réalité, les utilisateurs spécifient, le plus souvent, les données qu'ils souhaitent acquérir indépendamment des services qui fournissent ces données car il est plus facile de connaitre la donnée désirée plutôt que le service qu'elle la fournisse. De plus, pour une même donnée, il peut y avoir plusieurs services fournisseurs. Par conséquent, une tâche est considérée comme un objectif à réaliser qui est spécifié sous format orienté données. Par ailleurs, une tâche peut être spécifiée par un utilisateur de deux manières différentes : soit sous forme d'une *requête* explicite ou bien sous forme d'une *règle événementielle* capable d'être déclenchée par un certain événement.

5.2.4.1. Les requêtes utilisateur

Une requête est une tâche explicitement spécifiée par un utilisateur. Il s'agit d'une demande de service formulée par un utilisateur à un instant donné. Une requête *i* notée Req_i est définie par un couple ($User_i$; *G*) comme étant une association entre l'utilisateur *i* qui a spécifié la requête et l'objectif *G* (*GM; GC*) souhaité à travers cette requête.

5.2.4.2. Les règles événementielles

Un événement *E* (*Event*) se réfère à une situation particulière qui peut se produire dans un environnement ambiant tels que : une hausse inhabituelle de température, une présence d'un objet suspect, une fuite de gaz, etc. Un événement peut être détecté et signalé par un *service sensible aux événements*. Une *règle événementielle ER (Event Rule)* est définie comme étant un niveau intermédiaire entre les utilisateurs et les objectifs à réaliser quand un événement particulier se produit dans l'environnement. En conséquence, une *règle événementielle* ER_i est définie par un triplet ($User_i$; E_i ; *G*) comme étant une association entre l'événement particulier E_i et l'objectif *G* (*GM; GC*) que l'utilisateur $User_i$ souhaite atteindre comme réponse lorsque l'événement E_i se produit dans l'environnement ambiant. Nous constatons que les *règles événementielles* proposent un format plus générique qui englobe même les requêtes. De plus, ces règles permettent aux systèmes de devenir plus réactif à la détection des événements dans un environnement ambiant. Par conséquent, dans ce travail, on s'intéresse à la spécification des tâches sous formes des *règles événementielles*.

5.3. Présentation d'un Framework de composition de services

5.3.1. Principe général du Framework

Comme nous l'avons vu dans la **section 4.3.2 du chapitre 4**, la majorité des Frameworks proposés pour la composition de services souffrent de plusieurs lacunes quant à la prise en compte des caractéristiques et des exigences des environnements ambiants. Parmi ces lacunes, nous citons, entre autres, la supposition forte de certains Frameworks sur la nature d'un environnement ambiant. Ils stipulent que l'environnement est statique et l'invocation des services est déterministe. Cette forte hypothèse ne permet pas de tenir pleinement compte des caractéristiques des environnements ambiants et de répondre à leurs exigences. Dans ce travail, nous proposons un nouveau Framework qui vise la réalisation, dans un cadre orienté service, des exigences des systèmes intelligents ambiants détaillées dans la **section 1.4 du chapitre 1.** Il vise aussi à prendre en compte, au mieux, les caractéristiques des environnements ambiants qui posent des défis scientifiques considérables. Tous ces objectifs tournent autour d'un objectif principal qui consiste à réaliser des compositions de services qui soient sensibles aux événements. En d'autres termes, des compositions de services construites d'une manière réactive en guise de réponse à des détections d'événements dans un environnement ambiant. Nous convenons alors d'appeler ce nouveau Framework *FrEvASeC* (***Fr**amework for **Ev**ents **A**ware **Se**rvice **C**omposition). FrEvASeC* joue le rôle d'une plate-forme de médiation qui fournit quatre fonctions principales à savoir: le monitoring, la composition, la sélection et l'invocation des services. Toutes ces fonctionnalités peuvent être déclenchées automatiquement lorsqu'un éventuel événement se produit dans l'environnement ambiant. Ainsi, *FrEvASeC* est un Framework qui réagit à la détection des événements et informe les utilisateurs concernés après l'exécution des services appropriés. De plus, *FrEvASeC* met à jour les connaissances de l'environnement ambiant par des informations contextuelles de haut niveau résultant de l'exécution des services.

Le principe général du fonctionnement du système *FrEvASeC* est montré sur la **figure 5.7**. Tout d'abord, ce fonctionnement repose sur une séparation claire et nette entre les trois catégories de services définies précédemment à savoir : les *services sensibles aux événements*, les *services sensibles au contexte* et les *services de control des dispositifs*. Ensuite, tous ces services sont contrôlés par le cœur du système *FrEvASeC* qui est constitué des modules de monitoring, de composition, de sélection et d'invocation des services. Enfin, le système proposé est configurable par les utilisateurs et réactif aux événements. En effet, un utilisateur peut configurer certains paramètres du système et peut également recevoir des notifications en guise de retour du système. Globalement, ce fonctionnement passe par quatre étapes principales à savoir: une première étape des spécifications des utilisateurs, une deuxième étape de détection des événements, une troisième étape de gestion des événements détectés et une quatrième et dernière étape de notification des événements détectés.

Dans la première étape, un utilisateur peut spécifier les événements qu'il souhaite surveiller dans l'environnement ambiant. L'utilisateur doit aussi associer à ces événements des *règles événementielles* dans lesquelles, il indique les objectifs souhaités à atteindre dans le cas de leur apparition. L'utilisateur peut également indiquer certaines actions de control à

entreprendre en cas d'apparition d'un événement. En outre, l'utilisateur peut spécifier lors de cette première étape ses exigences en termes de qualité de service requise. Ces exigences de qualité peuvent être indiquées sous forme de pondération des différents paramètres de qualité de services. Dans la deuxième étape, lorsqu'un événement probable survient dans l'environnement, il sera détecté et rapporté par les *services sensibles aux événements*. L'événement détecté est aussitôt transmis au système à l'aide d'un mécanisme de souscription / notification. En effet, notre système s'inscrit auprès de tous les *services sensibles aux événements.* Ces derniers se chargent alors de notifier le système dés qu'un événement soit détecté dans l'environnement. Dans la troisième étape, le système tente de gérer l'événement détecté afin d'atteindre le but associé à cet événement. La gestion des événements est essentiellement basée sur le monitoring, la composition, la sélection et l'invocation des services. Dans la quatrième et dernière étape, l'ensemble ou une partie de l'objectif atteint par le système est transmis à l'utilisateur concerné sous forme d'une notification. À cette étape, le système peut éventuellement accomplir les actions de contrôle des dispositifs qui sont déjà spécifiées par l'utilisateur.

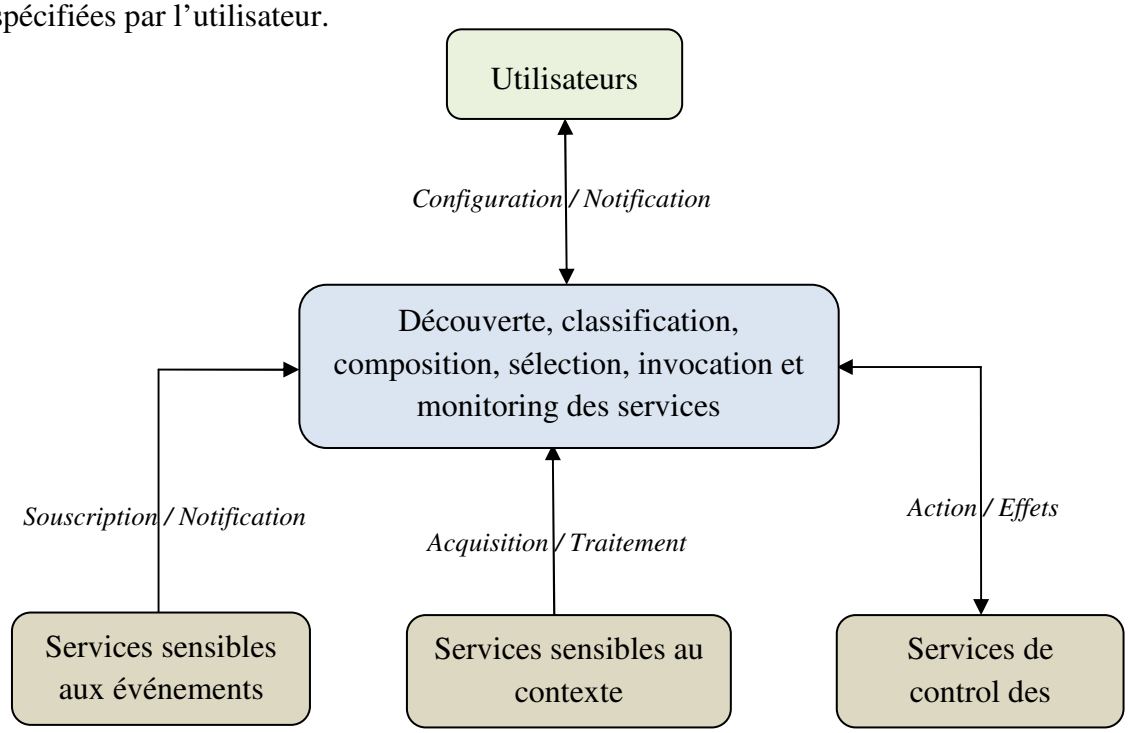


Figure 5.7 : Principe général du fonctionnement du Framework *FrEvASeC*

5.3.2. Quelques scénarii illustratifs

Afin d'illustrer le fonctionnement concret du Framework *FrEvASeC* dans un environnement ambiant, nous décrivons deux types de scénarios : le premier scénario porte sur le contrôle de la température ambiante et le deuxième scénario porte sur la surveillance d'une maison. Ces deux scénarios reposent sur l'utilisation des services suivants :

- Service sensible aux événements: *temperatureNotification, motionNotifier.*
- Service sensible au contexte (acquisition du contexte): *getTemperature, getVideo, getSound, getRfidTag.*
- Service sensible au contexte (traitement du contexte): *temperatureAnalysis, postureRecognition, soundRecognition, getName, personRecognition, alertMessage.*

- Service de control de dispositifs : *turnOnAirConditionnair, turnOffAirConditionnair.*

Un message *M1 <airConditionnairState>* est retourné par les deux services *turnOnAirConditionnair* et *turnOffAirConditionnair* pour indiquer l'état du climatiseur tandis qu'un message *M <suspectObjectAlert>* est retourné par le service pour indiquer l'existence d'un objet suspect ou pas. Par ailleurs, l'utilisateur du système, appelé *User*, spécifie les événements et les *règles événementielles* suivantes :

- Evénement *E1* : *temperature> 35°*
- Evénement *E2* : *temperature< 25°*
- Evénement *E3* : *(motion=true) and (08:00<time<16:00)*
- Régle *ER1*: *(User ; E1 ; G1)* avec G1 :*({M1} ; {M1.airConditionnairState =On})*
- Régle *ER2*: *(User ; E2 ; G2)* avec G2 :*({M1} ; {M1.airConditionnairState =Off})*
- Régle *ER3*: *(User ; E3 ; G3)* avec G3 :*({M2} ; {M2. suspectObjectAlert =True})*

5.3.2.1. Scenario1 : température ambiante

Le service *temperatureNotification* vérifie périodiquement la température de l'environnement. Supposant qu'à un instant donné, la température dépasse les 35°. Alors un événement de type *E1* est aussitôt transmis par le service *temperatureNotification* au système. Ce dernier active alors le service *getTemperature* pour échantillonner les températures de l'environnement et le service *temperatureAnalysis* afin d'analyser ces températures. Dans le cas ou le résultat de cette analyse confirme l'événement détecté alors le système tente de réaliser l'objectif spécifié par la règle *ER1* qui consiste à mettre l'état du climatiseur à « On ». Pour atteindre cet objectif, le système invoque le service *turnOnAirConditionnair* afin d'allumer le climatiseur et refroidir la salle. Un effet inverse se produira lorsque le service *temperatureNotification* détecte une température inférieure à 25°. Dans ce cas, un événement de type *E2* est envoyé par le service *temperatureNotification* au système. A l'instar du cas précédent, le système active les services *getTemperature* et *temperatureAnalysis* afin d'échantillonner et d'analyser respectivement la température. Dans le cas ou le résultat de cette analyse confirme l'événement détecté alors le système tente de réaliser l'objectif spécifié, cette fois-ci, par la règle *ER2* qui consiste à mettre l'état du climatiseur à « Off » en invoquant le service *turnOffAirConditionnair* afin de réchauffer un peu la salle. Dans les deux cas de figure illustrés par ce scénario, le système réalise deux compositions de services, et ce, afin d'atteindre les objectifs associés aux événements *E1* et *E2*. Les deux compositions de services obtenues sont montrées sur la **figure 5.8.** Ces deux compositions sont déclenchées respectivement par les événements *E1* et *E2* détectés par le service sensible aux événements *temperatureNotifier*.

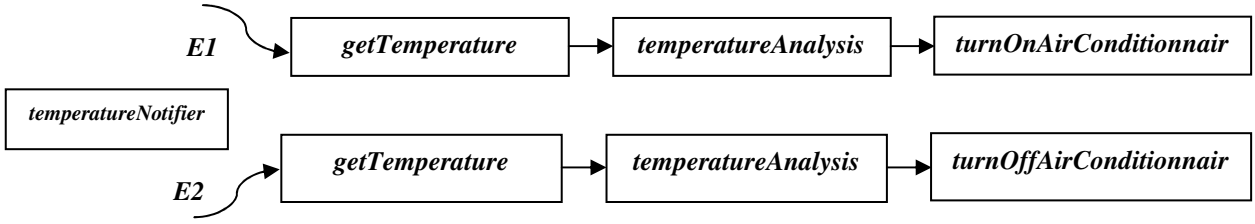


Figure 5.8 : Composition de services pour le control de la température ambiante

5.3.2.2. Scenario2 : mouvement suspect

Le service *motionNotifier* permet de surveiller une zone d'intérêt spécifiée sur le champ de vision d'une caméra. La zone d'intérêt est une zone particulière encadrée par un rectangle. Cette zone concerne généralement des coins ayant une fréquence de mouvement assez élevée, par exemple une porte, une fenêtre, etc. Le service *motionNotifier* permet ainsi d'observer périodiquement les mouvements dans la zone d'intérêt spécifiée et de remonter un événement de type *E3* en cas de détection d'un mouvement dans l'intervalle de temps *[8:00, 16 :00]*. En effet, dans cette période de temps, le propriétaire de la maison (*i.e. User*) est supposé être au travail. Donc la détection d'un mouvement en cette période de temps est considérée comme étant quelque chose de suspect. Supposant qu'à 12:00, un mouvement est détecté dans la zone d'intérêt. Alors un événement de type *E3* est aussitôt transmis par le service *motionNotifier* au système. Ce dernier active alors les services *getVideos*, *getSound* et *getRfidTag* afin d'identifier l'objet suspect. Tout d'abord, le service *getVideos* renvoie une séquence d'images (frames) JPEG contenant tous les objets détectés dans le champ de vision de la caméra. Ces images sont brutes et riches en termes d'information mais elles nécessitent un prétraitement afin de faciliter l'identification. Ainsi, le service *postureRecognition* utilise ces images afin d'identifier les personnes qu'elles contiennent en se basant sur la correspondance de l'allure des postures et des faces détectées avec celles enregistrées dans une base de donnée des personnes. Ensuite, le service *getSound* retourne un fichier audio du son enregistré. Ce dernier est utilisé par le service *soundRecognition* afin d'effectuer une reconnaissance vocale des personnes présentes. Ce mode d'identification est très utile, notamment lorsque la personne ne porte ni de capteurs ni de tag RFID. Enfin, le service *getRfidTag* effectue un scanne de tous les identifiants des tags afin de détecter les objets présents. Les identifiants détectés sont interprétés par le service *getName* qui retourne les noms des personnes et des objets correspondants à ces identifiants. Toutes les informations obtenues relatives aux différents modes d'identification sont analysées par le service *persons&ObjectsRecognition* afin, d'une part reconnaitre avec une plus grande certitude les objets et personnes présentes et d'autre part, estimer le niveau d'alerte sur la détection d'intrusion. Ainsi, la composition de services relative à ce scénario est montrée sur la **figure 5.9.** Cette composition est déclenchée par l'événement *E3* détecté par le service sensible aux événements *motionNotifier*.

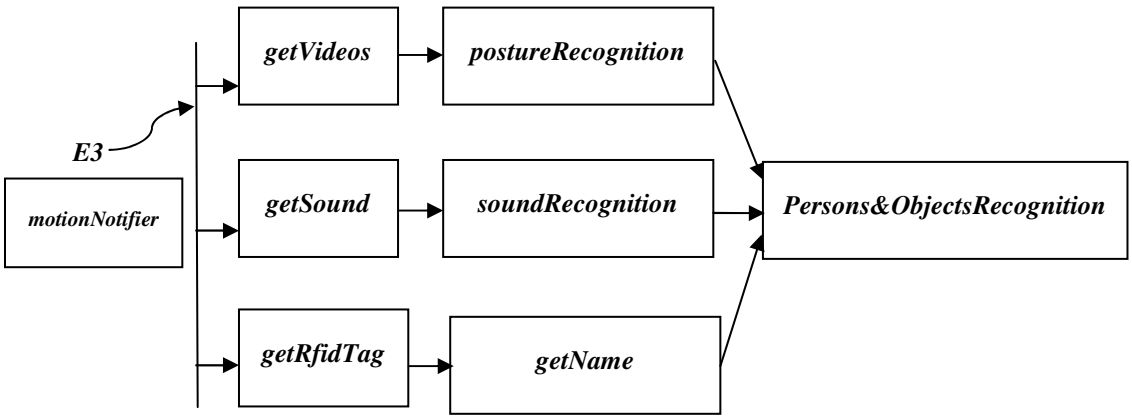


Figure 5.9: Composition de services pour l'analyse des mouvements suspects

5.3.3. Architecture détaillée du Framework

L'architecture détaillée du Framework *FrEvASeC* est illustrée sur la **figure 5.10**. Cette architecture est construite suivant un modèle multicouche afin de supporter un système adaptatif capable d'effectuer le monitoring, la composition, la sélection et l'invocation des services appropriés lorsqu'un événement survient dans un environnement ambiant. Ainsi, l'architecture de *FrEvASeC* est structurée en cinq couches principales à savoir : la *couche physique*, la *couche connectivité*, la *couche gestion des ressources*, la *couche services*, et la *couche contrôle*.

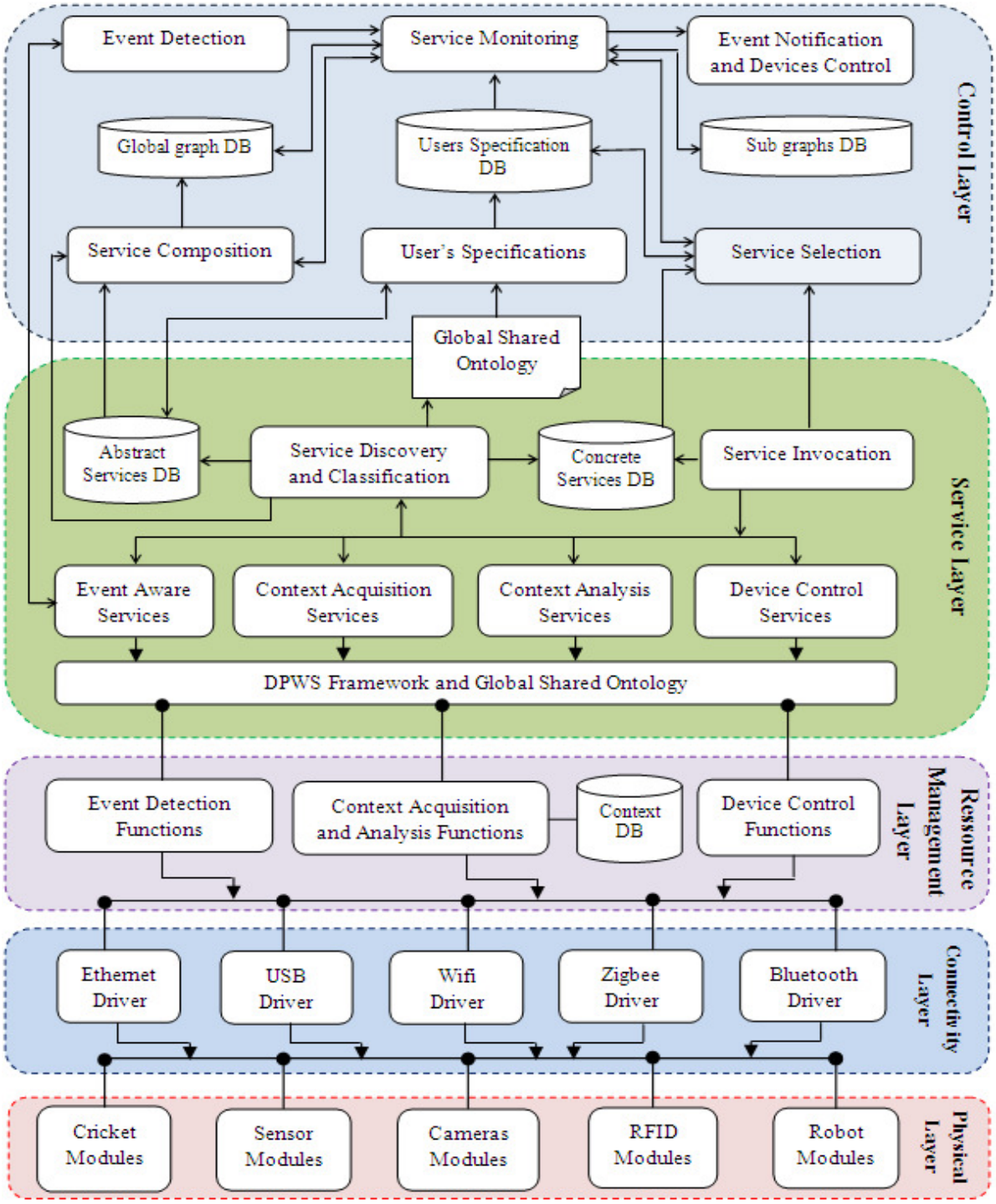


Figure 5.10 : Architecture détaillée du Framework *FrEvASeC*

5.3.3.1. La couche physique

La *couche physique* (*Physical Layer*) contient tous les dispositifs physiques intégrés dans l'environnement ambiant. Ces dispositifs sont hétérogènes et de nature très variés sur plusieurs aspects à savoir : les fonctionnalités offertes, leurs modes de programmation et de communication, leurs capacités de calcul, leur mobilité, etc. Ils présentent également une diversité très large allant de la simple lampe ou différents capteurs sensoriels, à des équipements plus complexes de type Robot, PDA, PC ou serveur. Chaque dispositif peut être fixe ou mobile dans l'environnement ambiant et peut intégrer plusieurs modules logiciels pour effectuer des tâches spécifiques. Par exemple, une caméra peut embarquer des modules d'acquisition et de transmission des flux vidéo tandis qu'un capteur peut embarquer des modules pour échantillonner et transmettre les différents paramètres ambiants (température, humidité, luminosité, etc.). Un module logiciel, lui même, doit être programmé suivant le langage de programmation supporté par le dispositif hôte. Par exemple, les modules pour les capteurs de type *Imote2* sont programmés avec le langage *NesC* alors que les modules pour le robot de type *Pekee II* sont en C#. Cet aspect exacerbe ainsi l'hétérogénéité des dispositifs disponibles au niveau de la *couche physique*.

5.3.3.2. La couche connectivité

La *couche connectivité* (*Connectivity Layer*) comprend les différents standards de communication supportés par les dispositifs de la *couche physique*. Cette couche gère toutes les connexions avec les dispositifs physiques afin de transmettre et de recevoir des données et des commandes. Plusieurs technologies filaires et sans fil sont cohabitées afin d'accéder au médium de communication. Ainsi, chaque dispositif peut être accessible via une ou plusieurs normes de communication filaires et/ou sans fil. Pour ce faire, un dispositif doit être équipé de plusieurs interfaces réseau pour pouvoir utiliser la totalité des réseaux d'accès déployés. Par exemple, une caméra de type *Axis* est accessible, d'une part via un réseau filaire utilisant IPv4 ou IPv6 et d'autre part via un réseau sans fil utilisant la norme WiFi. Par contre, une caméra de type *Logitech* est accessible uniquement via un port *USB*. Par ailleurs, la communication radio dans un réseau de capteurs de type *Imote2* est réalisée suivant la norme Zigbee. De plus, un ou plusieurs capteurs peuvent être connectés à l'ordinateur via un port *USB* pour agir comme des stations de base. Le rôle d'une telle station consiste à collecter des données en provenance du réseau de capteurs sans fil et les remonter vers l'ordinateur central pour d'éventuels traitements et visualisations. Pareil pour un lecteur RFID qui peut être connecté à un ordinateur via un port *USB*. Ce lecteur peut aussi établir des connexions avec les étiquettes (Tag) RFID situés à l'intérieur de son champ de lecture via la norme de communication Zigbee.

5.3.3.3. La couche gestion des ressources

La *couche gestion des ressources* (*Resource Management Layer*) est la première couche conçue pour masquer l'hétérogénéité des différents dispositifs physiques ainsi que leurs normes de communication. Cette couches fournie quatre types de fonctions à savoir: les *fonctions de contrôle des dispositifs*, les *fonctions d'acquisition du contexte*, les *fonctions d'analyse du contexte* et les *fonctions de détection des événements*. Les interfaces de ces fonctions sont fournies à la couche supérieure afin d'être utilisées dans la construction des

services de haut niveau. Tout d'abord, les *fonctions de contrôle des dispositifs* permettent de configurer et de commander les différents dispositifs de la *couche physique*. Par exemple, les fonctions qui permettent de configurer certains paramètres d'une caméra tels que le zoom et la résolution, tandis que d'autres fonctions comme *turnRight* et *turnLeft* contrôlent les actions de rotation de la caméra respectivement à droite et à gauche. Ensuite, les *fonctions d'acquisition du contexte* permettent l'interrogation des dispositifs physiques afin d'obtenir des données contextuelles telles que la température et l'humidité, tandis que les *fonctions d'analyse du contexte* permettent le traitement et l'analyse du contexte acquis. Enfin, les *fonctions de détection des événements* permettent de détecter les différents événements survenus dans l'environnement ambiant tels qu'une température ou une humidité anormales, un niveau d'énergie d'un certain dispositif est très faible, etc. La différence entre les *fonctions d'acquisition et de détection* est que l'acquisition se fait seulement sur une demande explicite, tandis que la détection dépend de l'apparition d'un événement. Par exemple, une température est acquise lors de l'interrogation des *fonctions d'acquisition* de la température. Par ailleurs, un seuil limite peut être spécifié pour une température ambiante. Lorsque ce seuil est dépassé, un événement relatif à une température anormale sera détecté et rapporté par une *fonction de détection d'événements*.

5.3.3.4. La couche services

Les fonctions fournies par la *couche gestion de ressources* sont développées suivant des langages de programmation différents et dépendent fortement des dispositifs qu'elles gèrent. De plus, l'accès à ces fonctions est restreint à leurs propriétaires. Il est alors nécessaire d'envelopper ces fonctions dans un modèle unifié permettant un accès publique et une utilisation à large échelle. Ainsi, la *couche services* (*Service Layer*) est la deuxième couche conçue pour cacher l'hétérogénéité qui se manifeste au niveau des couches inférieures, et ce, grâce au modèle orienté services. Les fonctions fournies par la couche inférieure sont encapsulées sous forme de services afin d'assurer un haut niveau d'abstraction et une accessibilité plus large. En effet, c'est au niveau de la *couche services* que nous trouvons les quatre catégories de services concrets décrites précédemment à savoir: les *services sensibles aux événements* (*Event-aware services*), les *services d'acquisition du contexte* (*Context acquisition services*), les *services de traitement du contexte* (*Context processing services*) et les *services de control de dispositifs (Devices control services)*. Premièrement, les *services sensibles aux événements* effectuent des souscriptions (abonnements) aux *fonctions de détection d'événements* afin de recevoir des notifications lorsqu'un des événements se produisent dans l'environnement ambiant. Deuxièmement, les *services d'acquisition du contexte* interrogent les *fonctions d'acquisition de contexte* afin d'obtenir un certain type de contexte. Troisièmement, les *services de traitement du contexte* analysent et interprètent les différentes données contextuelles en se basant sur les *fonctions d'analyse du contexte*. Quatrièmement, les *services de control de dispositifs* utilisent les *fonctions de contrôle des dispositifs* afin de commander les différents dispositifs disséminés dans l'environnement. Tous ces services peuvent être implémentés sous forme de services web et de services web pour dispositifs. Pour ces derniers, nous préconisons l'utilisation de la technologie DPWS (*Devices Profile for Web Services*). DPWS est une version récente de la technologie UPnP et qui est totalement compatible avec la pile des standards des services Web normalisés par le

consortium W3C (**section 2.3.2 du chapitre 2**). La technologie DPWS présente d'énormes avantages tels que : la réactivité à l'aide d'une communication événementielle, la découverte dynamique et décentralisée des services et le support de la mobilité des dispositifs. Par ailleurs, au niveau de la *couche services*, une ontologie globale partagée est définie pour associer une description sémantique pour tous les paramètres de contexte utilisés par les services afin de mieux comprendre leur signification et de faciliter leur réutilisation. Cette ontologie est décrite dans le langage OWL comme une hiérarchie de concepts. De plus, les services concrets sont dynamiquement découverts, mis à jour et classifiés dans des services abstraits par le *module de découverte et de classification des services (services discovery and classification module)*. Ensuite, ces services sont stockés dans deux types de bases de données: la base de données des services concrets (*CSDB: Concrete Services Data Base*) et la base de données des services abstraits (*ASDB : Abstract Services Data Base).* Enfin, les services concrets peuvent être invoqués grâce au *module d'invocation de services (services invocation module)*.

5.3.3.5. La couche contrôle

La *couche contrôle* (*Control Layer)* est chargée du control total de tout l'environnement ambiant en se basant sur six modules principaux à savoir : le *module des spécifications des utilisateurs* (*users's specification module),*le *module de détection des événements (event detection module),* le *module de monitoring des services (service monitoring module),* le *module de composition de services (service composition module),*le *module de sélection de services (service selection module),* le *module de notification et de contrôle des dispositifs (event notification and device control module).* Premièrement, le *module des spécifications des utilisateurs* constitue l'interface d'interaction entre le système et les utilisateurs. En effet, ce module permet aux utilisateurs de configurer le système en spécifiant plusieurs paramètres tels que : les règles événementielles, les préférences en terme de la qualité de service requise et éventuellement un ou plusieurs plan de services qui seront exécutés en cas d'occurrence de certains types d'événement. Deuxièmement, le *module de détection des événements* détecte et signale tous les événements qui surviennent dans l'environnement ambiant grâce à un mécanisme de communication de type souscription/notification avec tous les *services sensibles aux événements.* Troisièmement, le *module de monitoring des services* se charge d'atteindre le but associé à un événement détecté. Ce module appelle d'abord le *module de composition de services* afin de générer un plan (graphe) de services qui satisfait le but de l'événement détecté. Ensuite, le plan généré est exécuté en utilisant le *module de sélection de services* qui choisi les services à invoquer selon leur qualité. Enfin, les services choisis sont invoqués en utilisant le *module d'invocation de services*. Quatrièmement, le *module de notification et de contrôle des dispositifs* reçoit l'objectif atteint par le *module de monitoring des services* et se charge de notifier l'utilisateur concerné. Ce module peut éventuellement accomplir les actions de contrôle des dispositifs qui sont déjà spécifiées par l'utilisateur. Pour mener à bien ces différentes tâches, la *couche control* fourni un accès à trois types de base de données : la base des spécifications des utilisateurs (*USDB*), la base du graphe global (*GGDB :*) et la base des sous graphes (*SGDB*). Les techniques de construction du graphe global et des sous graphes seront détaillées dans le prochain chapitre.

5.4. Conclusion

Dans ce chapitre, nous avons décrit en détail notre modèle des connaissances. Ce modèle est très riche en termes d'informations car il englobe l'ensemble des connaissances manipulées dans un environnement ambiant. Ces informations concernent quatre entités principales à savoir : les services, les événements, le contexte et les utilisateurs. Tout d'abord, les services reflètent les fonctionnalités fournies par les différents dispositifs de l'environnement ambiant. Ainsi, nous avons distingué entre les services concrets qui implémentent réellement ces fonctionnalités et les services abstraits qui représente des classes de services concrets. L'abstraction des services permet un haut niveau de raisonnement sur les services indépendamment de leurs implémentations concrètes. La qualité de service est également prise en compte que ce soit dans son aspect statique ou bien dynamique. Ensuite, les événements sont modélisés comme des déclencheurs pour le système. En effet, à chaque type d'événement est associée une règle événementielle. Cette dernière est une requête implicite qui permet de guider le système vers les objectifs souhaités par l'utilisateur en cas d'apparition d'un événement. Puis, le contexte est modélisé dans sa forme simple par des paramètres de contexte et dans sa forme structurée par des messages de contexte. Ces messages sont produits et consommés par les différents services concrets. Enfin, les utilisateurs sont aussi pris en compte par le modèle des connaissances via leurs spécifications que ce soit en termes des préférences de qualité de service ou bien en termes de règles événementielles.

Le modèle des connaissances proposé constitue ainsi un support solide pour le Framework *FrEvASeC* en lui permettant une plus grande ouverture vers les différentes entités de l'environnement ambiant. Le modèle des connaissances permet également d'alimenter les différents modules des différentes couches du Framework *FrEvASeC* par des informations compréhensibles et exploitables. En effet, l'architecture du Framework *FrEvASeC* est construite suivant un modèle multicouche afin de supporter un système adaptatif et capable d'effectuer le monitoring, la composition, la sélection et l'invocation des services appropriés lorsqu'un événement survient dans un environnement ambiant. Plus précisément, cette architecture est structurée en cinq couches principales à savoir : la *couche physique*, la *couche connectivité*, la *couche gestion des ressources*, la *couche services*, et la *couche contrôle*.

Dans le travail de cette thèse, nous nous concentrons sur les deux couches supérieures (i.e. *couche services* et *couche contrôle*) du Framework *FrEvASeC*. Plus précisément, nous décrivons en détail les cinq modules principaux suivants: le *module de découverte et de classification des services*, le *module de composition de services*, le *module de sélection de services,* le *module d'invocation de services* et le *module de monitoring des services.* Ainsi, le principe de fonctionnement de tous ces modules fera l'objet du prochain chapitre.

CHAPITRE 6

Stratégie de composition de services

6.1. Introduction

Dans ce chapitre, nous présentons en détail notre stratégie globale de composition de services dans un environnement ambiant. Ce chapitre décrit les modèles orientés services sur lesquels reposent le fonctionnement et la collaboration des différents modules de notre Framework de composition de services présenté dans le chapitre précédent. Ces modèles, principalement au nombre de cinq, sont les suivants : modèle de découverte de services, modèle de classification de services, modèle de composition de services, modèle de sélection de services et enfin le modèle de monitoring de services. Ces déférents modèles fournissent des mécanismes de collaboration afin de réaliser la tâche de composition de services. Tout d'abord, les modèles de découverte et de classification de services se chargent respectivement de localiser et de préparer sémantiquement les services nécessaires à la composition de services. Cette préparation sémantique des services se base sur un modèle de *matching* sémantique qui se charge, dans un premier temps, d'établir une similarité sémantique entre les messages fournis et requis par les services concrets. Dans un second temps, il s'agit d'identifier les services qui sont fonctionnellement équivalents. Ensuite, le modèle de composition des services repose sur deux phases principales: la phase offline et la phase online. Dans la phase offline, un graphe global reliant tous les services abstraits disponibles est généré automatiquement en se basant sur des règles de décision sur les entrées et sorties des services. Dans la phase online, des sous graphes sont extraits spontanément à partir du graphe global selon les tâches à réaliser. Ces tâches peuvent être déclenchées soit par des événements détectés dans l'environnement ou bien par des requêtes soumises par des utilisateurs. Enfin, les modèles de sélection et de monitoring de services visent l'exécution concrète des graphes construits par le modèle de composition de services, et ce, dans un environnement dynamique et incertain toute en garantissant une meilleure qualité de service.

Ainsi, l'organisation de ce présent chapitre est scindée en quatre grandes parties. La première partie porte sur le modèle de découverte et de classification de services. Cette partie décrit d'abord l'approche de découverte de services, ensuite celle de classification de services. La deuxième partie porte sur la formalisation du modèle de composition de services. Ce modèle se base d'abord sur une représentation graphique des services abstraits. Ensuite, les graphes de composition de services sont construits grâce à des techniques de planification à base de règles, issues de l'intelligence artificielle (IA). Nous décrivons techniquement notre approche de composition de services par plusieurs algorithmes proposés. La troisième partie porte, quant à elle, sur le modèle de sélection de services qui se base sur trois phases principales : phase d'estimation de la qualité de service, phase d'invocation de services, et phase de sélection de services avec apprentissage. La quatrième et dernière partie porte sur le modèle de monitoring de services. Ce modèle se base sur plusieurs phases de monitoring de services afin de garantir une continuité de service malgré les pannes imprévisibles qui peuvent survenir dans l'environnement.

6.2. Modèles de découverte et de classification de services

Les services concrets sont découverts et classifiés dans des services abstraits par le module de découverte et de classification de services du Framework *FrEvASeC* (**section 5.3 du chapitre 5**). Tout d'abord, la découverte de services se charge de la recherche, la localisation et la sauvegarde des services concrets dans la base de services concrets (*CSDB*). Ensuite, la classification de services se charge de classifier les services concrets déjà découverts et disponibles dans la base *CSDB* dans des services abstraits. Ces derniers sont, eux mêmes, sauvegardés dans la base de services abstraits (*ASDB*). Ainsi, la classification de services est une phase complémentaire qui vient juste après la phase de découverte de services. Son rôle principal consiste à préparer les services abstraits nécessaires à la composition de services. Cette classification de services se base essentiellement sur l'ontologie globale partagée afin de comprendre la signification sémantique des différents paramètres utilisés par les messages d'entrés/sorties des services concrets. La découverte de services, quant à elle, son rôle principal consiste à mettre à jour dynamiquement et régulièrement la base de services concrets (*CSDB*) par les services qui apparaissent et disparaissent dans l'environnement ambiant.

6.2.1. Découverte de services

Dans un réseau local, un seul répertoire de service faisant office d'une base de données centrale de tous les services mis en ligne pourrait être approprié. Dans ce cas, la découverte de services peut être simplement effectuée par le gestionnaire de la base de données qui maintient les descriptions de tous les services disponibles. Toutefois, dans un environnement ambiant qui est, par définition, un réseau étendu à large échelle, une telle solution est peu envisageable. En effet, avec cette solution, tous les fournisseurs de services seront contraints d'enregistrer leurs services auprès d'un seul et unique répertoire de services. Ce dernier sera donc appeler à gérer un nombre important d'annonces de services effectuées par des fournisseurs de services géographiquement éloignés. Cela peut créer des gaulons d'étranglement à cause des accès simultanés au répertoire de services. Par conséquent, le temps d'attente pour annoncer ou rechercher un service augmente significativement. Par ailleurs, la publication de tous les services dans un seul répertoire ne permet pas d'assurer la redondance et la spécialisation (**section 2.5.2.2 du chapitre 2**). En effet, la redondance nécessite le déploiement de plusieurs répertoires identiques tandis que la spécialisation nécessite la séparation des répertoires de services déployés en fonction de leurs domaines spécifiques.

Afin de palier aux inconvénients d'une solution à un seul répertoire de service, notre approche repose sur une architecture utilisant plusieurs répertoires de services. Dans une telle architecture, un répertoire de services peut être spécifique à une zone géographique particulière (ex. réseau local) ou bien spécifique à un domaine particulier (ex. domaine médical). Il peut être également une copie d'un autre répertoire de service (redondance) afin d'augmenter la fiabilité des données. Chaque répertoire de services appartenant à cette architecture est appelé *Répertoire Local de Service* (*RLS*). Un RLS construit sa base de données à partir des messages d'annonce des fournisseurs locaux de services. En effet, chaque

nouveau service s'annonce (se publie) dans le registre local auquel il est rattaché. Ce rattachement peut être lié à une proximité en termes de zone géographique ou bien en termes de domaine d'intérêt. Tous les répertoires locaux sont eux même rattachés, via le module de découverte de services (**Figure 6.1**), à un registre global qui contient tous les services annoncés. Ce registre global n'est autre que la base de données des services concrets (CSDB). Ainsi, le rôle du module de découverte de services consiste à collecter et maintenir à jour tous les services disponibles dans l'environnement ambiant, i.e. déjà publiés dans les différents répertoires locaux. Comme le montre la **figure 6.1**, la relation entre le module de découverte de services et les répertoires locaux est régie par un mécanisme de souscription /notification (**section 2.5.4.2 du chapitre 2**).

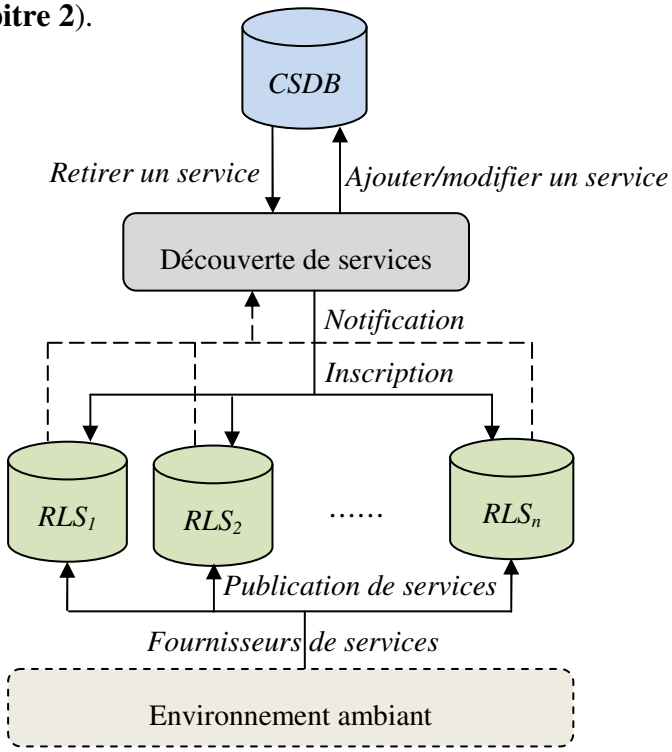


Figure 6.1 : Architecture de découverte de services

Dans un premier temps, le module de découverte de services effectue une souscription (abonnement) auprès de tous les répertoires locaux disponibles dans l'environnement ambiant. La souscription concerne trois types d'événements à savoir : un nouveau service est publié dans le répertoire, un service existant est retiré (supprimé) du répertoire ou la description d'un service existant est modifiée. Ces souscriptions sont gardées en mémoire de chaque répertoire de services comme des requêtes persistantes. Chaque répertoire de services envoie alors une réponse de manière asynchrone (notification) au module de découverte de services dès que l'un des trois événements précédents se produise. Il est à noter que lorsqu'il s'agit d'une publication d'un nouveau service, le répertoire local envoie la description WSDL du service publié ainsi que son identifiant. Par ailleurs, lorsqu'il s'agit d'une modification d'un service, le répertoire local envoie l'identifiant du service modifié ainsi que sa nouvelle description WSDL. Enfin, quand il s'agit d'une suppression d'un service, le répertoire local envoie juste l'identifiant du service à supprimer. Dans un second temps, le module de découverte de services se charge de rapporter toutes les mises à jour nécessaires au niveau de la base de données des services concrets (*CSDB*). Une mise à jour est effectuée sur la base *CSDB* dès

qu'une notification est reçue d'un répertoire local de services. Une fois la mise à jour est effectuée, le module de découverte de services informe le module de classification de services (**section 6.2.2**) afin de mettre à jours les classes de services déjà établies.

Notre module de découverte de services se base sur deux modes de découverte de services : mode actif et mode passif (**section 2.5.2.3 du chapitre 2**). En effet, lors du démarrage du système, le module de découverte de services initie la requête de découverte active pour connaitre la liste des répertoires locaux de services présents dans l'environnement ambiant. Cette requête de découverte est envoyée par annonce multicast ou broadcast (diffusion) sur le réseau. Les répertoires de services disponibles répondent, à leur tour, au module de découverte de services (unicast) en indiquant les informations pour les contacter (ex. adresse IP). Par la suite, le module de découverte de services effectue une souscription auprès des répertoires de services découverts pour écouter passivement les annonces spontanées relatives aux événements de publication, de modification ou de suppression de services. Par ailleurs, du fait de la possibilité de connexions et de déconnexions imprévisibles des répertoires de services, la requête active est renouvelée périodiquement par le module de découverte de services.

6.2.2. Classification de services

Après la phase de découverte de services décrite dans la section précédente, la base de services *CSDB* peut contenir un nombre considérable de services concrets puisque ces derniers sont découverts dans plusieurs répertoires de services déployés dans un réseau à large échelle. Ainsi, la possibilité d'avoir plusieurs services concrets qui offrent des fonctionnalités similaires est très élevée. Il est alors judicieux de procéder à une classification de ces services en termes de fonctionnalité fournies ou requises. Autrement dit, regrouper les services concrets qui fournissent les mêmes fonctionnalités dans une même classe de services. Cette classification permet de réduire considérablement le temps d'exploration des services concrets à la recherche de certaines fonctionnalités désirées. En effet, lorsqu'on désir rechercher une certaine fonctionnalité, il faut effectuer autant de recherches qu'il ya de services concrets qui fournissent cette fonctionnalité, et ce, dans le cas ou les services concrets ne sont pas classifiés. Contrairement, si les services concrets sont classifiés, une seule recherche suffit car désormais les fonctionnalités désirées seront recherchées dans des classes de services concrets plutôt que dans les services concrets eux-mêmes. Dans ce cas, une fois la classe recherchée soit trouvée, c'est tous les services concrets souhaités qui sont ainsi trouvés. Dans la **section 5.2.3.2 du chapitre 5**, nous avons défini les classes de services concrets fonctionnellement équivalents comme étant des services abstraits. Ainsi, une classe de services concrets est définie exactement par les mêmes paramètres définissant un service abstrait à savoir : une liste de messages d'entrée (*Inputs*), une liste de messages de sortie (*Outputs*), une catégorie de service et une liste de services concrets. Comme nous l'avons aussi souligné dans cette même section, l'équivalence fonctionnelle entre deux services concrets repose sur trois paramètres à savoir : les messages d'entrée, les messages de sorties et la catégorie de ces deux services. Ainsi, deux services concrets sont considérés comme fonctionnellement équivalents (similaires) si et seulement si ces deux services possèdent les mêmes messages d'entrée, les mêmes messages de sortie et appartiennent à une même catégorie de service. L'équivalence fonctionnelle entre deux services concrets est formalisée par la **définition 1** ci-dessous :

Définition1 (équivalence fonctionnelle) *: CS_i et CS_j deux services concrets fonctionnellement équivalentes si et seulement si ($CS_i.Category=CS_j.Category$) ∧ ($CS_i.Inputs=CS_j.Inputs$) ∧ ($CS_i.Outputs =CS_j.Outputs$).*

Ainsi, d'après cette définition, pour vérifier une équivalence fonctionnelle, il faut impérativement vérifier trois types d'équivalence : équivalence en termes de catégorie de services, équivalence en termes de messages d'entrée, et équivalence en termes de messages de sorties. L'équivalence en terme de catégorie est triviale car il suffit d'une simple comparaison entre les catégories de services qui sont au nombre de quatre : *Event-aware services, Devices control services, Context processing services,* et *Context acquisition services.* Cependant l'équivalence en termes de message d'entrées/sorties nécessite la recherche d'une similarité sémantique entre pairs de messages. Étant donné que les messages sont construits à partir des concepts, la recherche d'une similarité sémantique entre deux messages revient d'abord à rechercher une similarité sémantique entre leurs concepts respectifs. Pour ce faire, nous proposons un modèle de *matching* sémantique qui fourni une métrique pour mesurer la similarité sémantique entre deux concepts. Ce modèle de *matching* se base essentiellement sur une ontologie globale partagée qui décrit tous les concepts utilisés. Cette ontologie permet ainsi de comprendre la signification sémantique des différents concepts utilisés par les messages d'entrés/sorties des services concrets.

Comme nous l'avons souligné dans la **section 5.2.2.1 du chapitre 5**, un *concept Cpt* représente une information contextuelle qui est définie par un triplet <*nom, type, sémantique*> tels que : *nom* représente le nom syntaxique de *Cpt*, *type* représente le type syntaxique de *Cpt*, et *sémantique* représente la description sémantique de *Cpt* dans une ontologie. *Sémantique* lui même est représenté par un quadruplet <*O, Position, Pères, Fils*> tels que : *O* est l'ontologie de concepts à laquelle *Cpt* appartient, *Position* est l'emplacement (profondeur, largeur) du concept *Cpt* dans l'ontologie *O*, *Pères* est l'ensemble des concepts pères de *Cpt* et *Fils* l'ensemble des concepts *Fils* de *Cpt*. Dans l'ontologie *O*, le concept *Cpt* peut être soit un *concept terminal* soit un *concept intermédiaire*. *Cpt* est un *concept terminal* (feuille) s'il appartienne à une profondeur terminale d'une branche dans l'arborescence décrivant l'ontologie *O*. Inversement, *Cpt* est considéré comme un *concept intermédiaire* s'il ne se situe pas sur les feuilles de l'arborescence décrivant l'ontologie *O*. En se basant sur cette description d'un concept, nous définissons trois types de relations de *matching* sémantique entre deux concepts : *similitude*, *généralisation* et *restriction*.

- **Similitude** (≡) : *Deux concepts Cpt1 et Cpt2 sont similaires dans une ontologie O si et seulement si Cpt1 et Cpt2 possèdent le même type syntaxique et se réfèrent un même concept dans l'ontologie O.*

$$Cpt1 \equiv Cpt2 \overset{O}{\Leftrightarrow} (Cpt1.Type = Cpt2.Type) \wedge (Cpt1.Position = Cpt2.Position) \dots (1)$$

- **Généralisation** (≫) : *Le concept Cpt1 est une généralisation du (i.e. généralise le) concept Cpt2 dans une ontologie O si est seulement si Cpt1 figure parmi les pères de Cpt2 dans l'ontologie O.*

$$Cpt1 \gg Cpt2 \overset{O}{\Leftrightarrow} (Cpt1 \in Cpt2.Peres) \dots (2)$$

- **Restriction** (≪) : *Le concept Cpt1 est une restriction du (i.e. restreint le) concept Cpt2 dans une ontologie O si est seulement si Cpt1 figure parmi les fils de Cpt2 dans l'ontologie O.*

$$Cpt1 \ll Cpt2 \overset{O}{\Leftrightarrow} (Cpt1 \in Cpt2.Fils) \dots (3)$$

On constate que l'équivalence $\mathrm{Cpt1} \gg \mathrm{Cpt2} \overset{\mathrm{O}}{\Leftrightarrow} \mathrm{Cpt2} \ll \mathrm{Cpt1}$ est bien vérifiée. En effet, si un concept *Cpt1* généralise un autre concept *Cpt2*, alors *Cpt1* doit nécessairement appartenir aux *pères* du concept *Cpt2*. Donc *Cpt2* est forcement un *fils* de *Cp1*, i.e. une restriction de *Cpt1*. Maintenant que les relations qui puissent exister entre *concepts* sont connues, et sachant que les *messages* sont construits à partir d'un ensemble de *concepts*, il serait alors possible de déduire les relations entre les messages eux mêmes. Partant de ce principe, nous définissons deux relations essentielles entre les messages : *égalité* et *dérivée*.

- **Egalité** (=) : *Deux messages M1 et M2 sont égaux par rapport à une ontologie O si et seulement si M1 et M2 contiennent le même nombre de concepts, i.e. ont la même taille (cardinalité), et il existe une relation de similitude bijective entre les concepts de M1 et M2 dans l'ontologie O.*

$$\mathrm{M1} = \mathrm{M2} \overset{\mathrm{o}}{\Leftrightarrow} (\mathrm{M1.Cardinality} = \mathrm{M2.Cardinality}) \wedge (\forall\ \mathrm{Cpt1} \in \mathrm{M1}, \exists\ \mathrm{Cpt2} \in \mathrm{M2} \,/\, \mathrm{Cpt1} \equiv \mathrm{Cpt2}) \dots (4)$$

- **Dérivée** (∂) : *Lorsque deux messages M1 et M2 sont de même taille et les concepts du message M1 peuvent être obtenues par restriction des concepts du message M2 dans une ontologie O, alors le message M1est considéré comme une dérivée du message M2 par rapport à l'ontologie O.*

$$\mathrm{M1}\ \partial\ \mathrm{M2} \overset{\mathrm{o}}{\Leftrightarrow} (\mathrm{M1.Cardinality} = \mathrm{M2.Cardinality}) \wedge (\forall\ \mathrm{Cpt1} \in \mathrm{M1}, \exists\ \mathrm{Cpt2} \in \mathrm{M2} \,/\, (\mathrm{Cpt1} \equiv \mathrm{Cpt2}) \vee (\mathrm{Cpt1} \ll \mathrm{Cpt2})) \dots (5)$$

Nous distinguons deux types de dérivée, une dérivée intermédiaire et une dérivée terminale.

- **Dérivée intermédiaire** (∂I): *Une dérivée intermédiaire est une dérivée qui contient au moins un concept intermédiaire.*

$$\mathrm{M1}\ \partial\mathrm{I}\ \mathrm{M2} \overset{\mathrm{o}}{\Leftrightarrow} (\mathrm{M1.Cardinality} = \mathrm{M2.Cardinality}) \wedge (\exists!\ \mathrm{Cpt} \in \mathrm{M1} / \mathrm{Cpt\ est\ Intermediare})$$

$$\wedge (\forall\ \mathrm{Cpt1} \in \mathrm{M1}, \exists\ \mathrm{Cpt2} \in \mathrm{M2} \,/\, (\mathrm{Cpt1} \equiv \mathrm{Cpt2}) \vee (\mathrm{Cpt1} \ll \mathrm{Cpt2})) \dots (5.1)$$

- **Dérivée terminale** (∂T): *Une dérivée terminale est une dérivée dont tous les concepts sont des terminaux.*

$$\mathrm{M1}\ \partial\mathrm{T}\ \mathrm{M2} \overset{\mathrm{o}}{\Leftrightarrow} (\mathrm{M1.Cardinality} = \mathrm{M2.Cardinality}) \wedge (\forall\ \mathrm{Cpt1} \in \mathrm{M1}, (\mathrm{Cpt1\ est\ Terminal})$$

$$\wedge (\exists\ \mathrm{Cpt2} \in \mathrm{M2} \,/\, (\mathrm{Cpt1} \equiv \mathrm{Cpt2}) \vee (\mathrm{Cpt1} \ll \mathrm{Cpt2}))) \dots (5.2)$$

D'après la **formule 5** qui définie la dérivée d'un message, on constate que chaque message est soit une *dérivée terminale* ou bien une *dérivée intermédiaire* de lui même. En effet, les concepts d'un message sont similaires à eux mêmes. Comme nous l'avons décrit dans la **section 5.2.2.1 du chapitre 5**, les *concepts intermédiaires* existent uniquement au niveau sémantique et ne possède pas un type syntaxique. Dans la réalité, un *concept*

intermédiaire ne peut pas exister au tant que tel, mais son existence est incarnée par ses *fils* directs si ces derniers sont des *terminaux* sinon il est incarné par ses *fils* indirects qui sont des *terminaux*. Les *fils* indirects d'un *concept intermédiaire* sont obtenues par instanciation successives de concepts jusqu'à obtenir tous les concepts *fils* qui sont des *terminaux*. Ainsi, notre modèle de classification de services s'intéresse aux *concepts terminaux* (feuilles de l'ontologie) car ils représentent les concepts qui sont concrètement fournis par les services. A titre d'illustration, soit un service concret *CS* qui prend en entrée un message M contenant le concept *adresse*, notée *M < adresse >*. D'après l'ontologie relative au concept « adresse » décrite dans la **section 5.2.2.1 du chapitre 5**, le concept *adresse* est un *concept intermédiaire* et peut avoir trois concepts *fils terminaux* : *adresse physique, adresse mail ou adresse IP*. Ainsi, le message *M* peut avoir les trois formes *dérivées terminales* suivantes: *M1 < adresse physique >*, *M2 < adresse mail >*, *ou M3 < adresse IP >*. En effet, les trois messages *M1*, *M2* et *M3* représentent une instanciation concrète du message *M* qui ne peut pas exister en tant que tel dans la réalité. Donc, le service *CS* peut fonctionner correctement si on lui donne, comme entrée, l'un des trois messages suivants : *M1*, *M2* ou *M3*. Lorsqu'un message n'est pas une *dérivée terminale*, il faut alors dériver successivement ce message jusqu'à obtenir toutes ses *dérivées terminales* possibles.

La procédure complète de classification d'un service concret est détaillée par l'algorithme CSC (*Concrete Service Classification*) de la **figure 6.2**. L'algorithme *CSC* commence d'abord par la transformation (instanciation) de tous les messages présentés sous forme de *dérivés intermédiaires* à des messages sous forme de *dérivés terminales* (**ligne 4**). Cette instanciation est réalisée par la fonction *DerivateMessage* (**ligne 4**) qui prend en entrée l'ontologie *O* et le message *M* à dérivé. En sortie, cette fonction retourne toutes les *dérivées terminales* possibles du message *M* par rapport à l'ontologie *O*. Une fois que tous les *messages terminaux* d'un service concret soient obtenus, alors il est question de vérifier l'existence de ces messages dans la liste globale des messages appelée *ListOfMessages.* Cette liste maintien tous les *messages terminaux* déjà instanciés, et ce, afin d'éviter la duplication des messages. De plus, un nom de la forme « *Mi* » est attribué à chaque message de la liste *ListOfMessages* afin de garder des noms standards pour tous les messages disponibles et faciliter la comparaison entre les services. En effet, les services sont comparés grâce aux noms attribués à leurs messages d'entrées/sorties. Lors de la phase de vérification, si un message ne figure pas dans la liste *ListOfMessages* alors il sera renommé puis ajouté à cette liste (**ligne 15**). Dans le cas contraire, le message est juste renommé en lui attribuant le nom standard du message qui lui correspond dans la liste *ListOfMessages* (**ligne 11**). En effet, la vérification des messages se focalise sur la comparaison entre les structures (contenus) des messages et non pas sur leurs noms (**ligne 9**). Cela provient du fait que les noms des messages sont, en général, attribués par les fournisseurs de services concrets de différentes manières. Donc un message peut exister effectivement dans la liste *ListOfMessages* en termes de structure mais son nom n'est pas sous une forme standard. Dans ce cas de figure, il suffit juste de renommé ce message.

Les services concrets contenant des messages intermédiaires comme entrée/sortie possèdent forcement plusieurs instances possibles. Reprenant l'exemple précédent, le service concret *CS* qui prend en entrée le message *M <adresse>* possèdent trois instances possibles : instance *CS1* avec comme entrée le message *M1 <adresse physique>*, instance *CS2* avec

comme entrée le message *M2 <adresse mail>*, et instance *CS3* avec comme entrée le message *M3 <adresse IP>*. L'ensemble des instances possibles d'un service concret appelé *InstancesOfConcreteService* est obtenu par la fonction *instantiateConcreteSevice* (**ligne 23**) qui prend comme entrées le service concret désiré et la listes des *messages terminaux TerminalMessages* de ce service. Les instances d'un service concret appartiennent forcement à des classes de services différentes car elles possèdent des *messages terminaux* différents. Donc, un service concret possédant plusieurs instances appartient à plusieurs classes de services, i.e. services abstraits. Les instances d'un service concret sont regroupées selon le critère d'équivalence fonctionnelle énoncé par la **définition 1**.

A l'instar de la liste globale des messages, les classes de services sont aussi sauvegardées dans une liste de services abstraits nommée *ListOfAbstractServices*. Un nom standard de la forme « *ASi* » est attribué à chaque classe de service de cette liste. La classification proprement dite d'un service concret commence à la **ligne 24**. Chaque instance de la liste *InstancesOfConcreteService* est classifiée soit dans l'une des classes de services existante dans la liste *ListOfAbstractServices* ou bien dans une nouvelle classe de services. La comparaison avec les classes existante est réalisée selon le critère d'équivalence fonctionnelle énoncé par la **définition 1** et implémenté par la fonction *FunctionalEquivalence* (**ligne 27**) qui prend en entrée une instance d'un service concret et une classe de service et retourne un booléen qui indique si l'instance appartient à la classe ou non. Lorsqu'une instance n'appartient à aucune classe (**ligne 34**) alors une nouvelle classe est crée pour le service concret (**ligne 38**). Dans le cas ou l'instance appartient à une classe existante, alors il suffit d'ajouter le service concret à la liste des services concrets de cette classe (**ligne 29**). Afin de connaitre toutes les classes de services auxquelles un service concret appartient, l'algorithme CSC maintient une liste de classe appelée *ListOfClasse* pour chaque service concret.

La relation entre le module de classification de services et les autres modules est illustrée par la **figure 6.3**. Comme le montre cette figure, la classification de services se charge de regrouper tous les services concrets déjà découverts et disponibles dans la base CSDB dans des services abstraits. Ces derniers sont, eux mêmes, sauvegardés dans la base de services abstraits (ASDB). Ainsi, la classification de services est considérée comme une phase complémentaire, qui vient juste après la phase de découverte de services. Son rôle principal consiste donc à mettre en évidence toutes les fonctionnalités disponibles pour une éventuelle utilisation pour la composition de services. Cette classification de services utilise un *matching* sémantique essentiellement basé sur une ontologie globale.

La classification initiale des services concrets disponibles dans la base CSDB consiste à exécuter l'algorithme *CSC* de la **figure 6.2** sur chacun de ces services concrets. La disposition initiale des services concrets dans la base CSDB n'est pas importante. De même, les noms attribués aux messages dans la description initiale des services concrets n'est pas importante puisque notre modèle de classification se charge de leur donner des noms appropriés et plus compréhensibles. Une fois la classification initiale soit terminée, la classification d'un nouveau service concret et plus facile puisque les listes des classes et des messages sont déjà disponibles.

Concrete Service Classification (CSC)

Inputs: Concrete Service: CS, Ontology: O, ListOfMessages, ListOfAbstractServices

Output: ListeOfCalsses

Begin

```
1.  ListOfClasses=∅ ; TerminalMessages=∅ ;
2.  Messages = CS^in ∪ CS^out
3.  For each M in Messages do
4.    TerminalMessages= TerminalMessages∪ DerivateMessage (O, M)
5.  End for
6.  For each M in TerminalMessages do
7.     messageFound=False; i=1
8.     While (messageFound=False) and (i<=ListOfMessages.Length) do
9.        If (M. Structure= ListOfMessages[i].Structure) then
10.          messageFound =True
11.          M.Name= ListOfMessages[i].Name
12.       End if
13.       Increment i
14.    End while
15.    If (messageFound=False) then
16.       ListOfMessages.Length = ListOfMessages.Length +1
17.       i= ListOfMessages.Length
18.       ListOfMessages[i].Structure=M.Structure
19.       M.Name=M_i
20.       ListOfMessages[i].Name=M.Name
21.    End if
22. End For
23. InstancesOfConcreteService=instantiateConcreteSevice(CS, TerminalMessages)
24. For each instance in InstancesOfConcreteService do
25.    classeFound=False; i=1
26.    While (classeFound=false) and (i<=ListOfAbstractServices.Length) do
27.       If FunctionalEquivalence(instance, ListOfAbstractServices[i]) then
28.          classeFound =True
29.          ListOfAbstractServices[i].ConcretsServices.add(CS)
30.          ListOfClasses.add(ListOfAbstractServices[i].Name)
31.       End if
32.       Increment i
33.    End while
34.    If (classeFound=False) then
35.       ListOfAbstractServices.Length = ListOfAbstractServices.Length +1
36.       i= ListOfAbstractServices.Length
37.       ListOfAbstractServices[i].Name=AS_i
38.       ListOfAbstractServices[i]. ConcretsServices.add(CS)
39.       ListOfAbstractServices[i].Inputs=instance.Inputs
40.       ListOfAbstractServices[i].Outputs=instance.Outputs
41.       ListOfAbstractServices[i].Category=instance.Category
42.       ListOfClasses.add(ListOfAbstractServices[i].Name)
43.    End if
44. End For
```

End

Figure 6.2 : Algorithme de classification des services concrets

La suppression d'un service concret induit sa suppression dans toutes les classes auxquelles il appartient. Puisque l'algorithme *CSC* maintient une liste de classe pour chaque service concret (*ListOfClasse)*, donc il suffit de supprimer le service concret désiré de toutes ses classes spécifiées dans cette liste. Ainsi, un service abstrait (une classe de service) sera automatiquement supprimé s'il ne contient aucun service concret. Les opérations d'ajout et de suppression d'un service concret sont déclenchées par le module de découverte de services (**section 6.2.1**). En effet, le module de découverte de services se charge d'informer le module de classification de services sur l'apparition et la disparition de services afin de mettre à jours les classes de services déjà établies et éventuellement en créer d'autres. Il est à noter que dans les cas d'ajout ou de suppression d'une classe de services, le module de classification de services informe, à son tour, le module de composition de services afin de mettre à jour le graphe global de services (**section 6.3**).

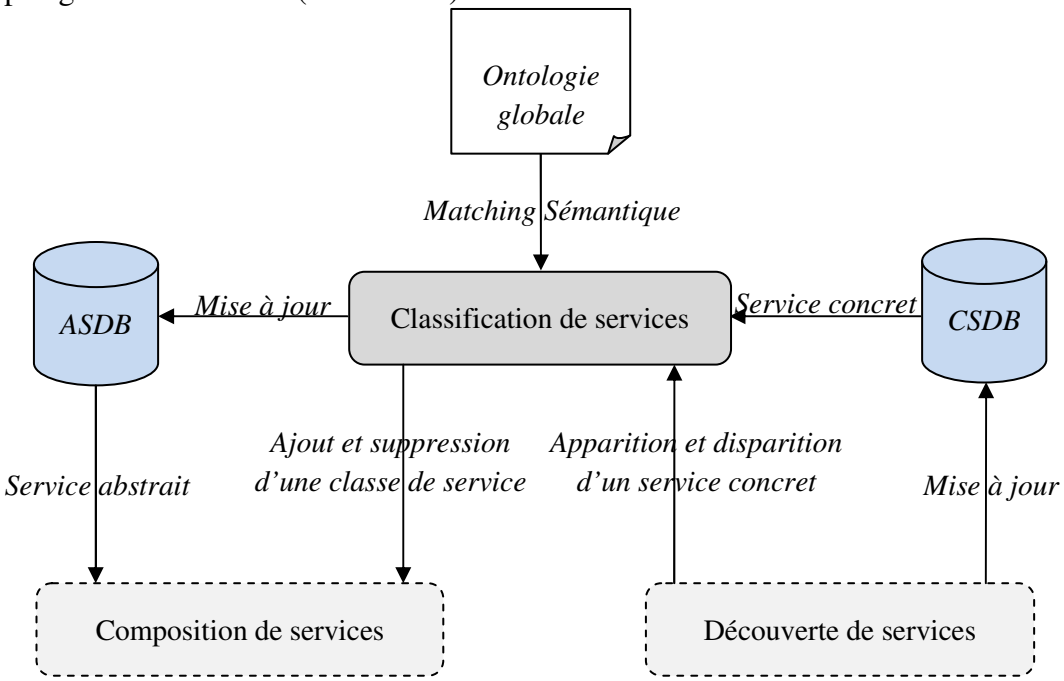


Figure 6.3 : Vue externe de la classification de services concrets

6.3. Modèle de composition de services

Nous avons vu dans les sections précédentes que les services concrets sont découverts et classifiés respectivement par les modules de découverte et de classification de services. Les classes de services (services abstraits) obtenues mettent en évidence toutes les fonctionnalités fournies ou requises par les services. Cependant, la classification de services ne décèle pas les relations fonctionnelles qui peuvent exister entre les différents services abstraits. En effet, les fonctionnalités requises par un service peuvent être fournies par un ou plusieurs services. De plus, les fonctionnalités fournies par un service peuvent être utilisées par un ou plusieurs services. Il est alors nécessaire, après la phase de découverte et de classification de services, d'effectuer une phase de rechercher de toutes les relations fonctionnelles qui puissent exister entre les services. Cette phase est réalisée par le module de composition de services qui se base sur les messages d'entrées/sorties des services abstraits. La composition de services est

définie par un graphe où chaque service abstrait est explicitement lié à ses prédécesseurs par ses entrées et à ses successeurs par ses sorties. Ainsi, la composition de services se focalise sur l'aspect externe d'un service abstrait afin d'étudier ses relations avec les autres services abstraits. En effet, pour la composition de services, un service abstrait est une boite noire qui fourni des entrées et des sorties. Ces dernières sont alors utilisées afin de créer des liaisons externes avec d'autres services abstraits. Ainsi, nous pouvons définir une composition de services comme étant un processus qui consiste à définir une combinaison appropriée (plan) des services abstraits.

6.3.1. Principe de composition de services

Notre approche de composition de service se base sur des règles logiques de raisonnement sur les entrées et les sorties des services abstraits (i.e. des classes de services qui sont fonctionnellement équivalents). Ces règles visent d'abord la création d'un graphe global de tous les services abstraits disponibles, ensuite l'extraction des sous graphes du graphe global afin de réaliser des tâches spécifiques. Le graphe global est construit automatiquement en reliant les entrées et les sorties de tous les services abstraits disponibles. Lors de cette construction, les services et les messages qui n'apportent aucune valeur ajoutée sont éliminés de sorte que le graphe résultant soit global et optimal en termes de fonctionnalités offertes. Le graphe résultant est global car il intègre toutes les fonctionnalités disponibles, et il est optimal car aucune fonctionnalité n'est dupliquée grâce à l'élimination des messages et des services identiques. Les sous graphes, quant à eux, sont extraits automatiquement du graphe global dans le but de répondre à des besoins spécifiques des tâches données. Ainsi, un sous graphe contient exactement les fonctionnalités nécessaires pour réaliser la tâche qui lui a été assignée. Cette approche de composition de service présente plusieurs avantages. D'abord, le graphe global peut être construit en offline avant que le système reçoit des tâches à réaliser. Cela permet d'augmenter significativement les performances du système notamment son temps de réponse à une tâche donnée. En effet, il sera plus rapide d'extraire un sous graphe pour réaliser une tâche en online dés lors que le graphe global est déjà en place. Ensuite, un sous graphe contient un ensemble de services abstraits qui peuvent avoir plusieurs réalisations concrètes. Ainsi, l'exécution d'un sous graphe est adaptée en fonction des services concrets disponibles au moment de cette exécution. Enfin, le graphe global assure la réalisation des tâches de point de vue fonctionnelle, sans tenir compte, à priori, des aspects non-fonctionnels relatifs à la qualité de service et au contexte d'utilisation. Puisque ces derniers sont sujets à des variations continuelles, il serait plus judicieux d'en tenir compte à des stades ultérieurs de composition notamment au moment de la réception d'une tâche et au moment de son exécution.

6.3.2. Représentation graphique d'une composition de services

6.3.2.1. Notations graphiques

Une composition de services est représentée par un graphe orienté *G (S, A)* tel que *S* est l'ensemble des services qui constituent les sommets du graphe *G* et *A* l'ensemble des messages qui constituent les arrêtes du graphe *G*. Soit *S1* et *S2* deux sommets du graphe *G*. Il existe une arrête qui relie le sommet *S1* au sommet *S2* si et seulement si il existe un message *M* qui est produit comme sortie par le service *S1* et qui peut être consommé comme entrée par

le service *S2*. Ainsi, la composition des deux services *S1* et *S2* est représentée par le graphe simple de la **figure 6.4** suivante :

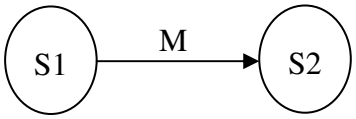


Figure 6.4 : Graphe simple de composition de services.

Dans un graphe de composition de services, un message *M* peut être produit et consommé par plusieurs services. Les services qui produisent le message *M* sont appelés *sources de M* et les services qui consomment le message *M* sont appelés *destination de M*. Plus formellement, nous définissons les sources et les destinations d'un message dans un graphe de composition de services de la manière suivante :

Définition2 (Sources d'un message) : *Les sources d'un message M dans un graphe de composition de services G, noté sources(M), est l'ensemble des services qui produisent le message M dans G, i.e. sources(M)= {S_i / ($S_i \in G$) et ($M \in S_i^{out}$)}*

Définition3 (Destinations d'un message) : *Les destinations d'un message M dans un graphe de composition de services G, noté destinations(M), est l'ensemble des services qui consomment le message M dans G, i.e. destinations(M)= {S_i / ($S_i \in G$) et ($M \in S_i^{in}$)}.*

Dans un graphe de composition de services quelconque, un message peut avoir plusieurs sources et plusieurs destinations. Dans l'exemple de la **figure 6.4**, *sources(M)= {S1}* et *destinations(M)= {S2}*.

Définition4 (Graphe exécutable): *Un graphe de composition de services G est dit exécutable si et seulement si, quelque soit un service S quelconque du graphe G alors tous les messages d'entrée de S sont produits dans G. Autrement dit:{($\forall S_i \in G$) $\wedge$ ($\forall M \in S_i^{in}$), sources (M) $\neq \varnothing$ }*

Considérant, par exemple, le graphe de composition de services *G* de la **figure 6.5** suivante:

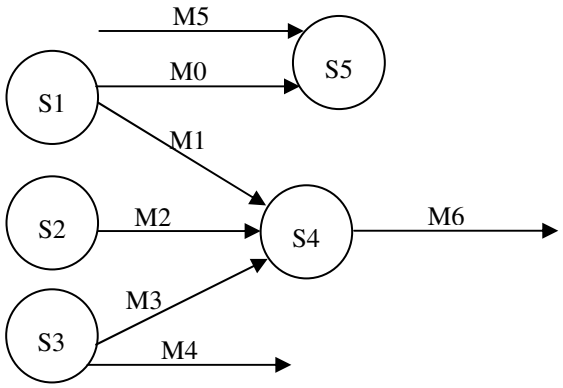


Figure 6.5 : Graphe de services non exécutable.

On constate que, sur le graphe *G* de la **figure 6.5,** toutes les entrées du service *S4* sont produites dans le graphe *G* malgré que sa sortie *M6* ne soit pas consommée par d'autres services dans le graphe *G*. De même, la sortie *M4* du service *S3* n'est pas aussi consommée

dans le graphe *G*. Par contre, le service *S5* possède une entrée *M5* qui n'est pas du tout produite dans le graphe *G*, i.e., *sources (M5)* =∅. Donc, d'après **la définition 4**, *G* n'est pas un graphe exécutable. Afin de le rendre exécutable, il suffit de supprimer du graphe *G* tous les services dont une ou plusieurs entrées ne sont pas produites. En supprimant le service *S5* du graphe *G*, on obtient le graphe de composition de services exécutable de la **figure 6.6** suivante :

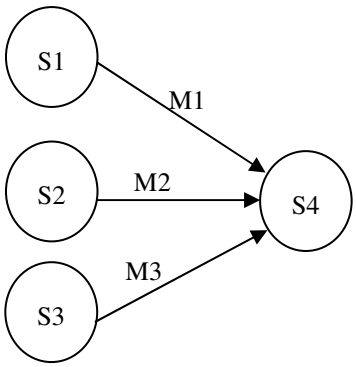


Figure 6.6 : Graphe de services exécutable.

Dans ce travail, on s'intéresse uniquement à la construction des graphes de composition de services exécutables, et ce, afin de garantir leurs exécution dans la phase de monitoring de services (**section 5**). Ainsi un graphe de composition de services exécutable *G* sera structuré en plusieurs niveaux de services. Chaque niveau du graphe *G* comporte l'ensemble des services qui peuvent s'exécuter en parallèle (simultanément). Par exemple, dans la **figure 6.6**, les services *S1*, *S2* et *S3* peuvent s'exécuter en parallèle, donc ils appartiennent au même niveau. Par contre, le service *S4* doit attendre la fin de l'exécution des services *S1*, *S2* et *S3* pour pouvoir s'exécuter. Donc, *S4* ne peut pas appartenir au même niveau des services *S1*, *S2* et *S3*. D'une façon générale, un graphe de composition de services *G* comporte trois types de structures à savoir: structure parallèle, structure séquentielle et structure de choix.

- ***Structure parallèle*** : deux services sont parallèles dans le graphe *G* s'ils appartiennent au même niveau du graphe *G*. Tous les services en parallèle peuvent s'exécuter simultanément. La **figure 6.7** montre un exemple de deux services *S1* et *S2* en structure parallèle dans un niveau *N*.

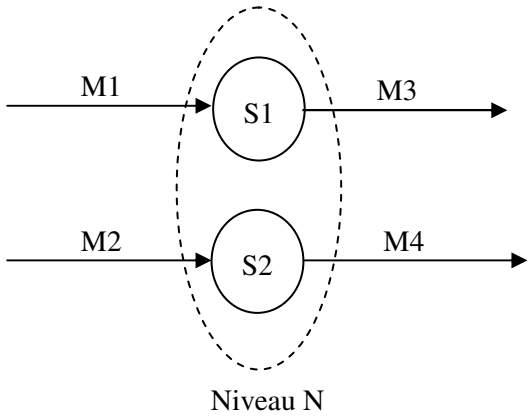


Figure 6.7 : Structure parallèle

- ***Structure séquentielle*** : deux services sont séquentiels dans le graphe *G* s'ils appartiennent à deux niveaux successifs du graphe *G*. Les services du niveau inférieur s'exécutent juste avant les services du niveau supérieur. La **figure 6.8** montre un exemple de trois services *S1*, *S2* et *S3* en structure séquentielle entre le niveau *N* et le niveau *N+1*. Dans ce cas de figure, *S3* est séquentiel avec (*S1* et *S2*).

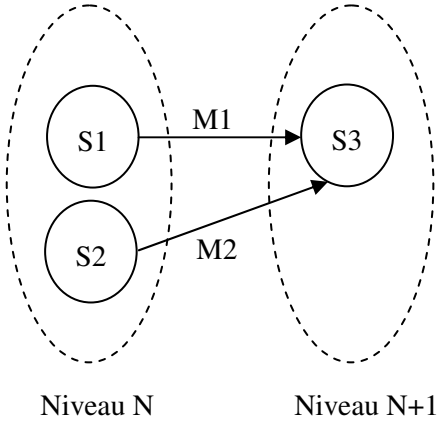


Figure 6.8 : Structure séquentielle

- ***Structure de choix*** : une structure de choix est relative à un message produit dans le graphe *G*. Deux services sont reliés dans une structure de choix par rapport à un message *M* dans le graphe *G* s'ils produisent le même message *M* dans *G*. Il existe alors autant de structures de choix entre deux services qu'il ya de messages de sortie en commun entre ces deux services. La **figure 6.9** montre un exemple de deux structures de choix par rapport aux messages *M1* et *M2*. Les services *S1*, *S2* et *S3* sont reliés à la structure de choix de *M1*, tandis que les services *S3* et *S4* sont reliés à la structure de choix de *M2*. On constate que *S3* participe simultanément à ces deux structures de choix.

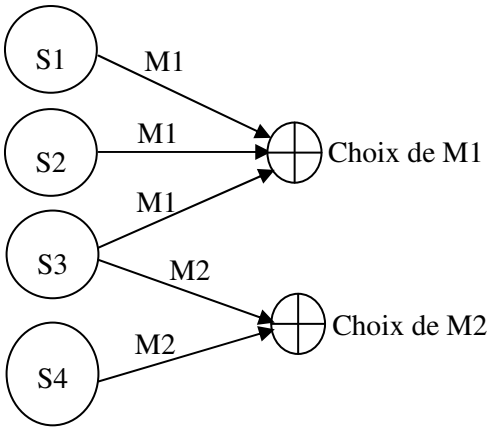


Figure 6.9: Structure de choix

6.3.2.2. Réduction des structures de choix

Un graphe de composition de services comportant des structures de choix est un graphe surchargé en termes de services à exécuter et de messages à échanger. En effet, une structure de choix nécessite l'exécution de plusieurs services afin de produire un même message. Ce dernier sera alors dupliqué sur le réseau autant de fois qu'il y a de services producteurs. Étant donné qu'un graphe de composition de services est un graphe exécutable, les structures de

choix peuvent alors provoquer des problèmes de control, de traitement et de communication lors de la phase d'exécution des services. (1) *Contrôle difficile*: les structures de choix nécessitent un contrôle particulier de chaque service exécuté et une synchronisation de la réception de tous les messages de choix. Cela implique alors un contrôle difficile des services producteurs et consommateur d'un message de choix. (2) *Surcharge des ressources de traitement* : les structures de choix nécessitent l'exécution d'un nombre important de services pour produire un même type de message. Chaque service exécuté exploite des ressources matérielles et logicielles. Cela surcharge alors ces ressources par des traitements inutiles. Et (3) *Surcharge des ressources de communication* : les structures de choix génèrent un flux important de données sur le réseau à cause de la duplication des messages de communication. Cela surcharge alors le réseau par l'échange des messages inutiles.

En particulier, dans un environnement ambiant, le contrôle se complique d'avantage vu le nombre important de services à exécuter et de messages à échanger. Supposant qu'il ya *n* services qui sont reliés à une structure de choix afin de produire un même message *m*. Lors de la phase d'exécution, il faut d'abord lancer l'exécution des *n* services en parallèle, contrôler chaque exécution et attendre la réception du message *m* de chaque service exécuté. Il faut ensuite synchroniser la réception du message *m* des *n* services producteurs avant de l'envoyer aux différents services consommateurs. Le système de contrôle dans ce cas sera cadencé par le service le plus lent. Cela provoque un temps d'attente considérable pour tous les services consommateur du message *m*. De plus, lorsque le nombre de services *n* devient significatif, le contrôle et la synchronisation de l'exécution des *n* services devient délicats voir impossible dans certains cas notamment lors des pannes de certains services. Admettant que parmi les *n* services lancer, il y a *k* services qui tombent en panne. Dans ce cas, le système de contrôle sera pris d'une part par la gestion des pannes des *k* services et d'autres parts par le contrôle des autres services en cours d'exécution. Cette charge importante sur le système peut alors provoquer une perte de contrôle de l'exécution des services. La tâche de contrôle se complique encore d'avantage si le système est appelé à gérer plusieurs structures de choix en même temps.

Afin de pallier à ces problèmes, l'idée consiste alors à réduire les structures de choix dans un graphe de composition de services avant même de passer à la phase d'exécution des services. La réduction d'une structure de choix d'un message *M* dans un graphe *G* consiste à maintenir une seule source pour le message *M* dans *G*. Cela revient alors à effectuer un choix entre tous les services producteurs du message *M*. Le résultat de la réduction d'une structure de choix est une structure séquentielle. La **figure 6.10** montre un exemple de réduction d'une structure de choix (à gauche) à l'une des deux structures séquentielles possibles (à droite).

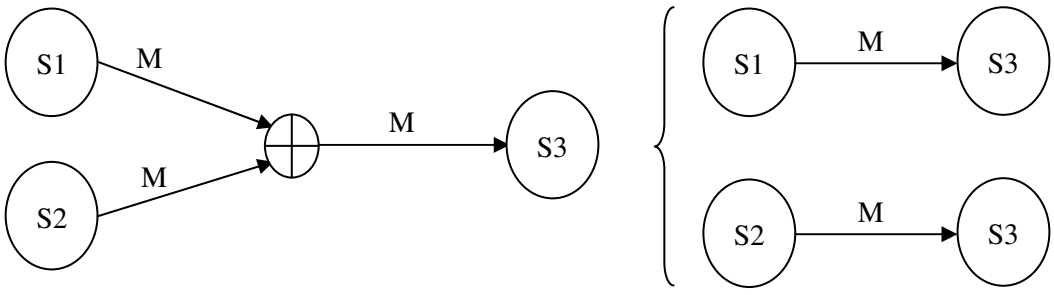


Figure 6.10 : Réduction d'une structure de choix

Ainsi, la réduction des structures de choix avant la phase d'exécution d'un graphe de composition de services permet de réduire considérablement le nombre de services et de messages à traiter par le système. Cela implique un contrôle plus facile de l'exécution des services et une diminution des ressources de traitements et de communication à exploiter. En réduisant toutes les structures de choix dans un graphe de composition de services, on obtient un graphe ne contenant que des structures parallèles et séquentielles.

Définition5 (Graphe de composition de services réduit) : *Un graphe de composition de services réduit G est un graphe de composition de services ne contenant ni de structures de choix ni de duplication des services et des messages. En d'autres termes, G est un graphe ne contenant que des structures parallèles et séquentielles de telle sorte que chaque service et message appartenant au graphe G ne figure qu'une seule fois dans ce graphe.*

Un graphe de composition de services réduit est structuré en plusieurs niveaux de services. Chaque niveau contient un ensemble de services qui peuvent s'exécuter simultanément (structure parallèle). Les services du niveau *N* s'exécutent après les services du niveau *N-1* et avant les services du niveau *N+1* (structure séquentielle). Par conséquent, un graphe de composition de services réduit possédant *N* niveaux de services peut être représenté par un graphe ayant la structure générale condensée montrée sur la **figure 6.11** suivante :

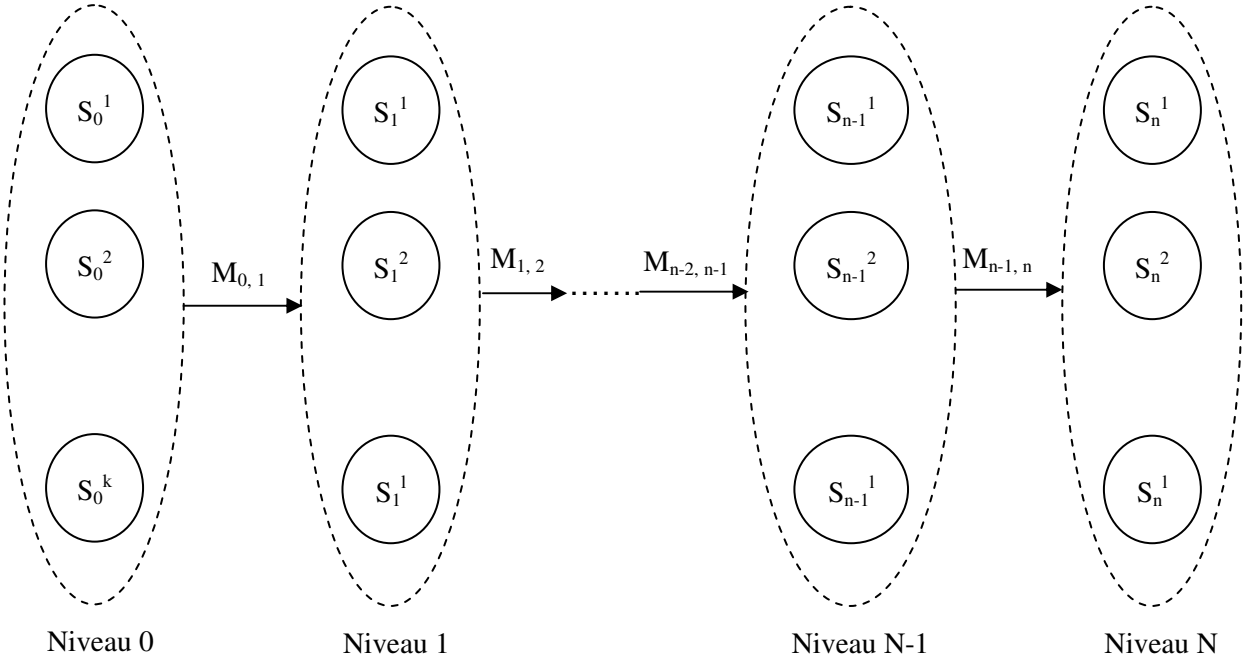


Figure 6.11 : Structure condensée d'un graphe de composition de services réduit

Sur la **figure 6.11**, S_i^j représente le service *i* du niveau *j*, tandis que $M_{i,\ i+1}$ représente l'ensemble des messages de sorties qui sont cumulés du niveau *0* jusqu'au niveau *i* tels que :

$$M_{0,1} = \bigcup_{S \in Niveau0} S^{out} \dots (6.1)$$

$$M_{i,i+1} = M_{i-1,i} \cup \bigcup_{S \in Niveau_i} S^{out} \ , \qquad 1 \leq i < n \dots (6.2)$$

Si on pose *Cardinality (M, G)* le nombre d'occurrence du message *M* dans le graphe *G* et *Cardinality (S, G)* le nombre d'occurrence du service *S* dans le graphe *G* alors :

$$(\forall \text{ un message } M \in G \,/\text{Cardinality}(M, G) = 1) \wedge (\forall \text{ un service } S \in G \,/\text{Cardinality}(S, G) = 1) \ldots (6.3)$$

La **formule 6.3** traduit le faite que chaque message apparaît une et une seule fois dans *G* et chaque service abstrait *S* apparaît une et une seule fois dans *G*.

Afin de construire un graphe de composition de services réduit *G*, il faut s'assurer d'abord que chaque service appartenant à *G* soit utilisé uniquement dans un seul niveau de *G*. Ensuite, il faut s'assurer que chaque message *M* fourni à un niveau *N* quelconque du graphe *G* soit un nouveau message. Autrement dit, le message *M* n'est pas déjà délivré dans les niveaux qui précédent le niveau *N* et ne sera délivré qu'une seule fois par les services de ce même niveau. Cela implique le maintient d'une seule source pour chaque message dans le graphe *G*. Pour ce faire, deux cas de figure sont à distinguer dans un graphe de composition de services :

- Toutes les sources (nœuds) sont complètement disjointes, i.e. $\forall S_i, S_j \in G, S_i^{out} \cap S_j^{out} = \emptyset$
- Le second cas étant la négation du premier cas, i.e. $\exists S_i, S_j \in G, S_i^{out} \cap S_j^{out} \neq \emptyset$

La construction d'un graphe réduit nécessite alors de tenir compte de chaque cas de figure.

Cas1 : toutes les sources sont disjointes

Lorsque toutes les sources sont disjointes dans un graphe de composition de services *G*, alors la **Propriété 1** énoncée ci-dessous est vérifiée dans *G*.

Propriété 1 *: Dans un graphe de composition de services G, chaque message m possède une et une seule source si et seulement si* $\forall S_i, S_j \in G, S_i^{out} \cap S_j^{out} = \emptyset$.

En effet, par définition d'une source d'un message *m* dans *G* (***Définition2***) *: sources(m)= {S_i/ ($S_i \in G$) et ($m \in S_i^{out}$)}.* Supposant d'abord que chaque message possède une et une seul source. Soit deux services quelconques S_i, S_j de *G* et soit un message *m* quelconque tel que $m \in S_i^{out}$ donc *sources(m)= {S_i}*. Si on a encore $m \in S_j^{out}$ donc *sources(m)= {S_i, S_j}*, cela signifie que m possèdent plusieurs sources (contredit la supposition). Donc $\forall S_i, S_j \in G\ S_i^{out} \cap S_j^{out} = \emptyset$.

Montrer l'autre sens de l'équivalence, c'est-à-dire : si $\forall S_i, S_j \in G\ S_i^{out} \cap S_j^{out} = \emptyset$ alors chaque message *m* possède une et une seule source, revient à montrer que : s'il existe au moins un message *m* qui possède plusieurs sources alors $\exists S_i, S_j \in G\ S_i^{out} \cap S_j^{out} \neq \emptyset$. Soit un message *m* et supposant que *m* possède plusieurs sources : Alors $\exists S_i, S_j \in G\ (m \in S_i^{out})$ et $(m \in S_j^{out})$ donc $S_i^{out} \cap S_j^{out} \neq \emptyset$ puisqu'au moins le message *m* est en commun entre les deux services S_i et S_j. Cela signifie que : $\exists S_i, S_j \in G\ S_i^{out} \cap S_j^{out} \neq \emptyset$. D'ou la preuve de la **Propriété 1**.

Lorsque toutes les sources sont disjointes, la construction d'un graphe de composition de services réduit est triviale. En effet, chaque message possède par défaut une et une seule source, d'après la **Propriété 1**. Dans ce cas, il suffit d'affecter chaque message à sa propre source qui est unique dans le graphe de composition de services.

Cas2 : il existe au moins deux sources non disjointes

En se basant sur la **propriété 1**, nous déduisons que: si $\exists S_i, S_j \in G\ S_i^{out} \cap S_j^{out} \neq \emptyset$ alors il existe au moins un message *m* dans *G* qui possède plusieurs sources. Dans ce cas de figure, il faut choisir une seule source pour le message *m*. Le choix des sources doit se faire de telle sorte que le graphe résultant soit minimal en termes de services retenus. Nous savons aussi que : si $\exists S_i, S_j \in G\ S_i^{out} \cap S_j^{out} \neq \emptyset$, alors on peut inférer les trois cas possible suivants:

Cas 2.1 *: $(S_i^{out} = S_j^{out})$*

Cas 2.2 *: $((S_i^{out} \subset S_j^{out}) \wedge (\neg(S_i^{out} \supset S_j^{out}))) \vee ((\neg(S_i^{out} \subset S_j^{out})) \wedge (S_i^{out} \supset S_j^{out}))$*

Cas 2.3 *: $(\exists m1/m1 \in (S_i^{out} \cap S_j^{out})) \wedge (\exists m2/(m2 \in S_i^{out}) \wedge (m2 \notin S_j^{out})) \wedge (\exists m3/(m3 \notin S_i^{out}) \wedge (m3 \in S_j^{out}))$*

Les différents cas de figures illustrés précédemment sont traduits en termes de règles de composition de services qui seront décrites dans la section suivante.

6.3.3. Règles de composition de services

Les règles de composition de services traduisent et implémentent techniquement les différents cas de figures qui puissent exister entre les services abstraits i.e. les classes de services. Ces règles reposent sur le principe de la création de liaisons dynamiques entre les entrées/sorties des services abstraits. Pour ce faire, ces règles partent de l'idée initiale que le graphe de composition de services réduit n'existe pas encore et la seule chose dont on dispose c'est les services abstraits disponibles. Par la suite, ces règles visent la construction d'un graphe global de tous ces services disponibles par la création de liaisons appropriées entre leurs messages d'entrées/sorties. Enfin, les règles de composition de services permettent de procéder à l'extraction des sous graphes du graphe global afin de réaliser des tâches spécifiques. L'extraction des sous graphes est généralement réalisée lorsqu'on souhaite identifier, dans le graphe global, juste les services qui contribuent effectivement à la réalisation d'un objectif bien précis. Les règles de composition de services proposées sont au nombre de cinq: *Règle de Marquage (Rule of Marking), Règle d'Égalité (Rule of Equality), Règle d'Inclusion Simple (Rule of Simple Inclusion), Règle d'Inclusion Complexe (Rule of Complex Inclusion)* et enfin *Règle de Démarquage (Rule of Unmarking).* Avant de détailler ces règles, nous proposons d'abord les quatre définitions ci-dessous qui sont nécessaires à leur fonctionnement.

Définition6 (Marquage d'un message) *: le processus de marquage d'un message m dans un graphe de composition de services G consiste à associé une et une seule source pour le message m dans le graphe G.*

Définition7 (Démarquage d'un message) *: le processus de démarquage d'un message m dans un graphe de composition de services G consiste à retirer (supprimer) la source pour le message m dans le graphe G.*

Définition8 (Table de marquage) *: la table de marquage est une liste de tous les messages produits dans un graphe de composition de services avec leurs sources respectives. Cette liste contient un ensemble de couples de la forme (message, service) tels que service et la source de message.*

Définition9 (Cardinalité de marquage d'un service)*: la cardinalité de marquage d'un service S est le nombre d'occurrence du service S dans la table de marquage.*

6.3.3.1. Règle de Marquage

La *règle de marquage* (*Rule of Marking*) consiste à marquer tous les messages de sortie des services abstraits appartenant à un niveau quelconque d'un graphe de composition de services. Ce processus de marquage (**définition 6**) conserve une seule *source* pour chaque message qui participe à une composition de services. La *règle de marquage* est illustrée par l'algorithme de la **figure 6.12**. Cet algorithme prend en entrée un niveau *AS* d'un graphe de composition de services représenté par un ensemble de services abstraits *(Set of abstract services).* Tous les messages produits au niveau *AS* sont sauvegardés dans la *table de marquage* (**définition 8**) avec leurs *sources* correspondantes. La procédure *checkMarkedMessages (m)* à la **ligne 3** vérifie l'existence du message *m* dans la *table de marquage*. Si le message *m* existe dans cette table, cela signifie que *m* est déjà produit (marqué), alors il n'est pas nécessaire de marquer *m* à nouveau. Contrairement, si *m* n'est pas trouvé dans la *table de marquage*, il sera marqué par la procédure *markeMessage (m, as)* à la **ligne 4**. Cette procédure ajoute le message *m* à la *table de marquage* et définit sa *source* comme étant le service abstrait *as*. La *cardinalité de marquage* (**définition 9**) du service *as* (*cardinalityOfMarking(as)*) est également incrémentée de 1 à la **ligne 5**. La *cardinalité de marquage* du service *as* reflète la valeur ajoutée (nombre de messages de sorties utilisés) du service *as* dans le graphe de composition de services. En agissant de cette manière sur chaque niveau d'un graphe de composition de services, la *règle de marquage* représentée dans la **figure 6.12** permet d'attribuer une et une seule *source* pour chaque message utilisé.

Rule of Marking

Input: Set of abstract services: AS

Begin

```
1.  For each abstract service as in AS do
2.    For each message m ∈ as^out do
3.      If (checkMarkedMessages(m)=false) then
4.        markeMessage(m, as)
5.        Increment cardinalityOfMarking(as)
6.      End if
7.    End for
8.  End for
```

End

Figure 6.12 : Règle de marquage

A l'issue de l'application de la *règle de marquage* sur un niveau *N*, la *table de marquage* conserve une seule source pour chaque message. Puisque un message n'est pas délivré par plusieurs sources alors il ne sera pas dupliqué. Cependant, la *règle de marquage* marque directement la première source trouvée pour un message *m* et elle ne tient pas compte des autres services qui peuvent aussi produire le message *m*. Ce marquage reste alors valable si on est en présence du premier cas de figure (**Cas1**) qui stipule que toutes les sources sont

disjointes, i.e. $\forall S_i, S_j \in G, S_i^{out} \cap S_j^{out} = \emptyset$. En effet, dans ce cas de figure, chaque service constitue la source de ses propres messages de sorties. Ainsi, chaque message possède par défaut une et une seule source dans le graphe *G* (**Propriété1**), et ce, quelque soit l'ordre de marquage.

6.3.3.2. Règle d'Égalité

La *règle d'égalité (Rule of Equality)* traite le cas de figure **Cas 2.1** relatif à une égalité parfaite entre les sorties de deux services. Cette règle permet de choisir un seul service parmi un ensemble de services qui apportent la même valeur ajoutée (même nombre de messages de sortie) à une composition de services. Soit par exemple un service *S1* qui possède deux messages de sortie *M1*et *M2,* et un service *S2* qui possède ces mêmes messages de sortie. Si on marque, par la *règle de marquage*, le service *S1* comme étant la source du message *M1* et le service *S2* comme étant la source du message *M2* dans un graphe de composition de services *G,* alors les deux services *S1* et *S2* seront maintenus inutilement dans le graphe *G*. En effet, dans ce cas de figure, il suffit de maintenir soit le service *S1* ou bien le service *S2* dans le graphe G sans perte de fonctionnalités. La *règle d'égalité* est illustrée par l'algorithme de la **figure 6.13**. Cet algorithme stipule qu'en cas d'égalité de sorties de deux services alors il faut garder dans le graphe de composition de services uniquement le service le plus grand en termes de nombre de message d'entrée. Ce choix est motivé par le faite qu'un service qui possède plusieurs messages d'entrée aura plus de services prédécesseurs dans le graphe de composition de services. Ainsi, le maintien de ce service permettra d'analyser simultanémenent plus de message de contexte dans le graphe de composition de services. Cela peut alors contribuer à améliorer la qualité des données des sorties des services. Dans le cas où les deux services possèdent le même nombre de messages en entrée, la *règle d'égalité* supprime le service qui contient le moins de services concrets. En effet, le service qui contient plus de services concrets offre plus d'alternatives de choix lors de la phase d'exécution. Dans le cas où les deux services présentent le même nombre d'entrées et contiennent le même nombre de services concrets, il suffit de maintenir l'un d'entre eux.

Rule of Equality

Input: Set of abstract services: AS

Begin

1. **For** each $as_i \in AS$ **do**
2. **If** $(\exists as_j \in AS/(j \neq i) \wedge as_i^{out}=as_j^{out}) \wedge ((|as_i^{in}|<|as_j^{in}|) \vee (|as_i^{in}|=|as_j^{in}|) \wedge (|as_i|<|as_j|)) \vee (|as_i^{in}|=|as_j^{in}|) \wedge (|as_i|=|as_j|) \wedge (i>j))$ **then**
3. Delete as_i from AS
4. **End if**
5. **End for**

End

Figure 6.13 : Règle d'égalité

6.3.3.3. Règle d'Inclusion Simple

La *règle d'inclusion simple (Rule of Simple Inclusion)* traite le cas de figure **Cas 2.2** relatif à une inclusion directe entre les sorties de deux services. Lorsque toutes les sorties d'un service *S1* sont complètement incluses dans les sorties d'un autre service *S2*, alors on est certain que toutes les fonctionnalités fournies par *S1* sont également fournies par *S2* et le contraire n'est pas vrai. Dans ce cas, le maintient du service *S1* dans un graphe de composition de services *G* est inutile, car *S1* n'apporte aucune valeur ajoutée (nouvelle fonctionnalité) au graphe *G*. Par contre, le maintient du service *S2* dans le graphe *G* est nécessaire car il apporte des nouveau messages qui ne sont pas fournis par le service *S1*. Ainsi, la *règle d'inclusion simple* consiste à maintenir uniquement les services ayant une certaine valeur ajoutée dans un graphe de composition de services. Cette règle, illustrée par l'algorithme de la **figure 6.14**, stipule que le service dont les sorties sont incluses dans les sorties d'un autre service devrait être supprimé du graphe de composition de services car il n'apporte aucune valeur ajoutée à ce graphe.

Rule of Simple Inclusion

Input: Set of abstract services: AS

Begin

1. **For** each $as_i \in AS$ **do**
2. **If** $(\exists\ as_j \in AS\ /\ (j \neq i) \wedge (as_i^{out} \subset as_j^{out}))$ **then**
3. Delete as_i from AS
4. **End if**
5. **End for**

End

Figure 6.14 : Règle d'inclusion simple

6.3.3.4. Règle d'Inclusion Complexe

Le cas de figure **Cas 2.3** présente un cas plus général dans lequel les sorties de deux services quelconques *S1* et *S2* ne sont ni en situation d'égalité ni en situation d'inclusion simple, mais elles ont juste certains messages en commun. Dans ce cas, *S1* et *S2* doivent être maintenus à la fois dans un graphe de composition de services *G*, puisque chacun de ces deux services apporte des nouveaux messages au graphe *G* que l'autre service n'apporte pas. Donc, la suppression de l'un de ces services implique une perte certaine de fonctionnalités dans le graphe *G*. Cependant, en analysant, par exemple, les messages de sortie du service *S1*qui ne sont pas en commun avec le service *S2*, il peut y avoir des inclusions simples avec d'autres services du même niveau ou bien des niveaux précédents du graphe *G*. Si cela est vérifié, alors les sorties du service *S1* sont incluses dans l'union des sorties de plusieurs autres services appartenant au graphe *G*. Dans ce cas de figure, le service *S1* peut être retiré du graphe *G* puisqu'il n'apporte réellement aucune valeur ajouté à ce graphe. Pour traiter ce cas, nous proposons une règle appelé *règle d'inclusion complexe* qui est illustrée par l'algorithme de la **figure 6.15**. Cet algorithme utilise la procédure *cheekMarkedMessages(m)* qui se base sur la *table de marquage* afin de connaitre si le message *m* est déjà marqué ou non. En effet, les messages qui sont déjà produits dans un graphe de composition de services seront automatiquement ajoutés à la *table de marquage* à l'issue de l'application de la *règle de*

marquage. L'algorithme de la **figure 6.15** vérifié également l'existence du message *m* dans les sorties des services appartenant à un même niveau du graphe de composition de services. Ainsi, la *règle d'inclusion complexe* stipule que, à chaque niveau *AS* d'un graphe de composition de services, un service sera supprimé si chaque message de sa sortie est déjà produit (i.e. existe dans *la table de marquage*) ou bien est inclus dans l'une des sorties des autres services du même niveau *AS*.

Rule of Complex Inclusion

Input: Set of abstract services: AS

Begin

1. **For** each $as_i \in AS$ **do**
2. **If** ($\forall m \in as_i^{out}$ / ((cheekMarkedMessages(m)=true) $\vee$ ($\exists as_j \in AS/(j \neq i) \wedge (m \in as_j^{out})$)) **then**
3. Delete as_i from AS
4. **End if**
5. **End for**

End

Figure 6.15: Règle d'inclusion complexe

6.3.3.5. Règle de Démarquage

L'objectif de l'application des quatre règles précédentes étant l'obtention d'un graphe de composition de services *réduit*. À l'issue de l'application de ces quatre règles sur chaque niveau d'un graphe de composition de services, il ne restera plus de structures de choix dans le graphe résultant. En effet, tous les choix des *sources* des messages sont effectués de telle sorte que les messages maintenus dans le graphe résultant ne possèdent qu'une seule occurrence. Le graphe résultant est dit « global » car il maintient toutes les fonctionnalités (messages) nécessaires fournies par les services. En d'autres termes, le graphe global résultant de l'application des quatre premières règles est un graphe orienté fonctionnalité. Cependant, lorsqu'on veut réaliser un objectif bien précis, certaines fonctionnalités du graphe global peuvent être nécessaires à la réalisation de cette objectif, d'autres peuvent être complètement inutiles. Cela nous mène ainsi à la définition de la notion de *sous graphe orienté objectif*.

Définition10 (Sous graphe orienté objectif)*: Un sous graphe orienté objectif est une partie du graphe global qui maintient uniquement les fonctionnalités nécessaires à la réalisation d'un objectif bien précis.*

L'objectif de la *règle de démarquage (Rule of Unmarking)* consiste à extraire à partir d'un niveau quelconque du graphe global les services qui contribuent effectivement à la réalisation d'un certain objectif. Concrètement, cela revient d'abord à supprimer les messages fournis dans le graphe global mais qui ne sont pas nécessaires pour atteindre l'objectif spécifié. Puisque ces messages sont déjà marqués dans *la table de marquage* par *la règle de marquage* alors il suffit de les supprimer de cette table. Ensuite, il faut décrémenter *la cardinalité de marquage* de tous les services qui fournissent les messages supprimés. Enfin, tous les services

dont la *cardinalité de marquage* est nulle sont considérés comme inutiles à la réalisation de l'objectif spécifié. Par conséquent, ces services peuvent êtres supprimés du graphe global. Ainsi, le résultat de l'application de la *règle de démarquage* est un *sous graphe orienté objectif.* La *règle de démarquage* est illustrée par l'algorithme de la **figure 6.16**. Cet algorithme agit sur chaque niveau *AS* du graphe global de la manière suivante: soit un ensemble de messages buts appelé *Goal*, et un service *as* appartenant à l'ensemble des services *AS*. Pour chaque messages *m* pour lequel le service *as* est marqué comme étant la *source* dans la *table de marquage*, si *m* n'est pas inclus dans l'objectif *Goal*, alors *m* est supprimé de *la table de marquage*. De plus, la *cardinalité de marquage* du service *as* (*cardinalityOfMarking(as))* est décrémentée de 1. Par conséquent, le service *as* sera automatiquement supprimé du graphe global si *as* n'est pas marqué comme étant *source* d'aucun messages dans la *table de marquage*, i.e. *cardinalityOfMarking (as) = 0*.

Rule of Unmarking

Input: Set of abstract services: AS, Goal: G

Begin

```
1.  For each as ∈ AS do
2.    For each m such as Source (m) = {as} do
3.     If (m ∉ G) then
4.       Delete m and Source(m) from marking table
5.       Decrement cardinalityOfMarking(as)
6.     End if
7.    End for
8.    If (cardinalityOfMarking(as) = 0) then
9.       Delete as from AS
10.   Else
11.      G=G U as_in
12.   End if
13. End for
```

End

Figure 6.16 : Règle d'inclusion complexe

6.3.4. Création automatique des graphes de composition de services

L'objectif de cette section consiste à montrer comment les règles de composition de services précédentes peuvent êtres exploitées afin de créer concrètement des graphes de composition de services. Il s'agit d'établir l'ordre d'application de ces règles et les structures de données adéquates pour leur fonctionnement. Il s'agit aussi de montrer l'utilisation de ces règles pour la construction d'un graphe globale et l'extraction des sous graphes de composition de services.

6.3.4.1. Construction d'un graphe global de composition de services

Nous appelons *AGoSC (Abstract Graph of Service Composition)* un graphe de composition de services *réduit* global dont tous les sommets sont des services abstraits. Afin de construire d'une manière automatique le graphe *AGoSC*, nous proposons un algorithme

offline appelé *AGoSC Construction Algorithm* illustré sur la **figure 6.17.** Cet algorithme consiste à relier d'une manière appropriée les entrées et sorties de tous les services abstraits disponibles en utilisant les quatre règles définies précédemment à savoir: la *règle de marquage, la règle d'égalité, la règle d'inclusion simple et la règle d'inclusion complexe*. Ainsi, cet algorithme prend en entrée l'ensemble des services abstrait *AS* qui se trouve dans la base des services abstraits (*ASDB*) et il retourne, comme sortie, un graphe global de composition de services (*AGoSC*). La méthode de construction consiste à construire le graphe *AGoSC* niveau par niveau en commençant du niveau initial jusqu'au niveau final. Ainsi, L'algorithme proposé agit comme un planificateur représenté par une paire *(AS, satisfiedAbstractServices)* tels que: *AS* est l'ensemble de tous les services abstraits disponibles et *satisfiedAbstractServices* contient tous les services abstraits satisfaits au niveau actuel en cours de construction appelé *abstractGraphLevel*. Ce dernier est un niveau abstrait défini par son identifiant *ID* et l'ensemble des services qu'il contient. Un service est considéré comme satisfaits si tous ses messages d'entrée sont disponibles. Cette condition nous garanti ainsi la construction d'un graphe de composition de services *exécutable* (**définition4**).

Initialement, tous les services appartenant à la catégorie des services d'acquisition de contexte (*context aquisition services*) sont considérées comme satisfaits. En effet, cette catégorie de service se présente sous la forme *getContext* et ne nécessite pas des messages d'entrées explicites. Donc, le premier niveau du graphe *AGoSC* est construit de tous les services d'acquisition de contexte (**lignes 3 et 4**). Par la suite, un nouveau niveau abstrait *abstractGraphLevel* est construit à chaque itération de l'algorithme de la **figure 6.17**. Une itération commence à la **ligne 7** et consiste à appliquer les quatre règles précédentes sur l'ensemble *satisfiedAbstractServices* afin de maintenir seulement les services appropriés. L'ordre d'application de ces règles est très important. La *règle d'égalité* et la *règle d'inclusion simple* sont vérifiées en premier lieu afin d'éliminer les cas triviaux et de réduire l'ensemble *satisfiedAbstractServices*. En second lieu, la *règle d'inclusion complexe* détecte et élimine les cas non triviaux. Enfin, la *règle de marquage* est appliquée afin d'associer une seule *source* pour les messages de sortie des services maintenus par les trois premières règles. Ainsi, l'ensemble *satisfiedAbstractServices* obtenus sera ajouté au niveau *abstractGraphLevel.* Ce niveau, lui même, sera ajouté au graphe global *AGoSC* qui est en phase de construction. Dans la même itération, l'ensemble *AS* est mis à jour en supprimant tous les services qui sont déjà satisfaits afin d'éviter les répétitions lors des prochaines itérations. Comme le nombre de services abstraits disponibles est fini, la mise à jour de *AS* garanti la terminaison de l'algorithme. Un nouvel ensemble de services satisfaits sera construit en utilisant la procédure *checkSatisfiedMessages* à la **ligne 22**. Cette procédure vérifie l'existence des messages d'entrées des services appartenant à *AS* dans la *table de marquage*. Les services dont toutes les entrées se trouvent dans la *table de marquage* seront ajoutés à l'ensemble *satisfiedAbstractServices*. Cet ensemble constitue ainsi l'entrée de la prochaine itération de l'algorithme de construction du graphe global. En conséquence, l'algorithme se termine lorsqu'il n'y a plus de services satisfaits, i.e. *satisfiedAbstractServices* $=\varnothing$.

AGoSC Construction Algorithm

Inputs: Set of abstract services: AS

Outputs: Abstract Graph of Services Composition: AGoSC

Begin

```
1.  AGoSC=∅, satisfiedAbstractServices =∅ , graphLevelID=0
2.  For each abstract service as in AS do
3.    If (as.getCategory.equals("ContextAcquisitionService"))
4.        satisfiedAbstractServices.add(as);
5.    End if
6.  End for
7.  While (satisfiedAbstractServices≠∅) do
8.    For each abstract service as in satisfiedAbstractServices do
9.      Remove as from AS;
10.   End for
11.   ruleOfSimpleInclusion(satisfiedAbstractServices);
12.   ruleOfEquality(satisfiedAbstractServices);
13.   ruleOfComlexInclusion(satisfiedAbstractServices);
14.   ruleOfMarking(satisfiedAbstractServices);
15.   If (satisfiedAbstractServices≠∅)
16.     abstractGraphLevel.ID=graphLevelID
17.     abstractGraphLevel.Service= satisfiedAbstractServices
18.     AGoSC.add(abstractGraphLevel);
19.     Increment graphLevelID;
20.     satisfiedAbstractServices=∅;
21.     For each abstract service as in AS do
22.       If(cheekSatisfiedMessages(as.getInput()))
23.         satisfiedAbstractServices.add(as);
24.       End if
25.     End for
26.   End if
27. End while
```

End

Figure 6.17 : Algorithme de construction d'un graphe global

6.3.4.2. Extraction de sous-graphes de composition de services

Nous appelons *SubGoSC (Sub Graph of Service Composition)* un sous graphe du graphe *AGoSC* qui est *orienté objectif*. Afin d'extraire d'une manière automatique le graphe *SubGoSC* du graphe *AGoSC*, nous proposons un algorithme *online* appelé *SubGoSC Extraction Algorithm* illustré sur la **figure 6.18.** Cet algorithme prend en entrée le graphe global *AGoSC* et un objectif spécifié *Goal* et il retourne, comme sortie, un sous graphe *SubGoSC*. L'objectif *Goal* n'est autre qu'un ensemble de messages qu'on désir obtenir. L'algorithme proposé se base sur une technique orientée objectif (*goal-driven technique*) utilisant la *règle de démarquage*. Cet algorithme d'extraction explore tous les niveaux du graphe *AGoSC* en commençant du dernier niveau jusqu'au premier niveau par un retour arrière progressif. A chaque niveau, la *règle de démarquage* est appliquée afin de ne

conserver que les services dont au moins un message de sortie appartient à l'objectif spécifié *Goal*. Ce dernier est également mis à jour par les entrées des services maintenus à chaque niveau. En procédant de cette manière, un sous-graphe sera extrait par la recherche, dans le graphe *AGoSC*, de tous les services prédécesseurs des services qui offrent un ou plusieurs messages de l'objectif initialement spécifié.

SubGoSC Extraction Algorithm

Inputs: Goal: G, Abstract Graph of Services Composition: AGoSC

Outputs: Sub Graph of Services Composition: SubGoSC

Begin

1. SubGoSC=∅, CurrentAbstractGraphLevelID=Last level ID in AGoSC;
2. **While**(CurrentAbstractGraphLevelID ≥ 0) **do**
3. CurrentServices=*AGoSC*.getServicesOfLevel(CurrentAbstractGraphLevelID);
4. ruleOfUnmarking(CurrentServices, G)
5. **If** (CurrentServices≠∅)
6. extractedAbstractGraphLevel.ID= CurrentAbstractGraphLevelID
7. extractedAbstractGraphLevel.Service= CurrentServices
8. SubGoSC.add(extractedAbstractGraphLevel);
9. **End if**
10. Decrement CurrentAbstractGraphLevelID;
11. **End while**
12. Inverse the level positions in SubGoSC

End

Figure 6.18 : Algorithme d'extraction d'un sous graphe

L'algorithme d'extraction des sous graphes est un algorithme *online* car il nécessite la spécification de l'objectif *Goal* avant de lancer l'extraction. Concrètement, cet objectif peut être spécifié par une requête de l'utilisateur ou bien dans une règle événementielle (**section 5.2.4 du chapitre 5**). Dans ce dernier cas, l'extraction d'un sous graphe *SubGoSC* est effectuée lorsqu'un événement est détecté. Le sous graphe résultant est considéré comme un graphe de composition de services qui satisfait la règle de l'événement détecté. Contrairement à l'algorithme d'extraction des sous graphes, l'algorithme de construction du graphe global est un algorithme *offline*. En effet, l'algorithme de construction d'un graphe global nécessite uniquement la base des services abstraits et peut être lancé, à n'importe quel moment, indépendamment de la spécification d'un quelconque objectif.

6.4. Modèle de sélection de services

6.4.1. Principe de sélection de services

La sélection de services consiste à choisir un service concret parmi un ensemble de services concrets fonctionnellement équivalent, i.e. appartenant à un même service abstrait. Contrairement à la composition de services qui se base sur une vision externe des services abstraits, la sélection de services se base sur une vision interne des services abstraits. En effet, pour la composition de services, un service abstrait est une boite noire qui fourni des entrées et des sorties. Ces dernières sont alors utilisées afin de créer des liaisons externes avec

d'autres services abstraits. La sélection de services, quant à elle, s'intéresse de prés au contenu interne des services abstraits. En d'autres termes, elle se focalise sur la manière de choisir un service concret à l'intérieur d'un service abstrait. Ce choix se base sur des paramètres non fonctionnels (qualité de service), et ce, contrairement à la composition de service qui se base sur des paramètres fonctionnels (entrées/sorties). Toutefois, la sélection de services est une étape complémentaire et nécessaire à la composition de services. En effet, comme le montre la **figure 6.19**, la sélection de services opère sur un graphe de composition de services (plan abstrait) déjà établi afin de le rendre exécutable (plan concret).

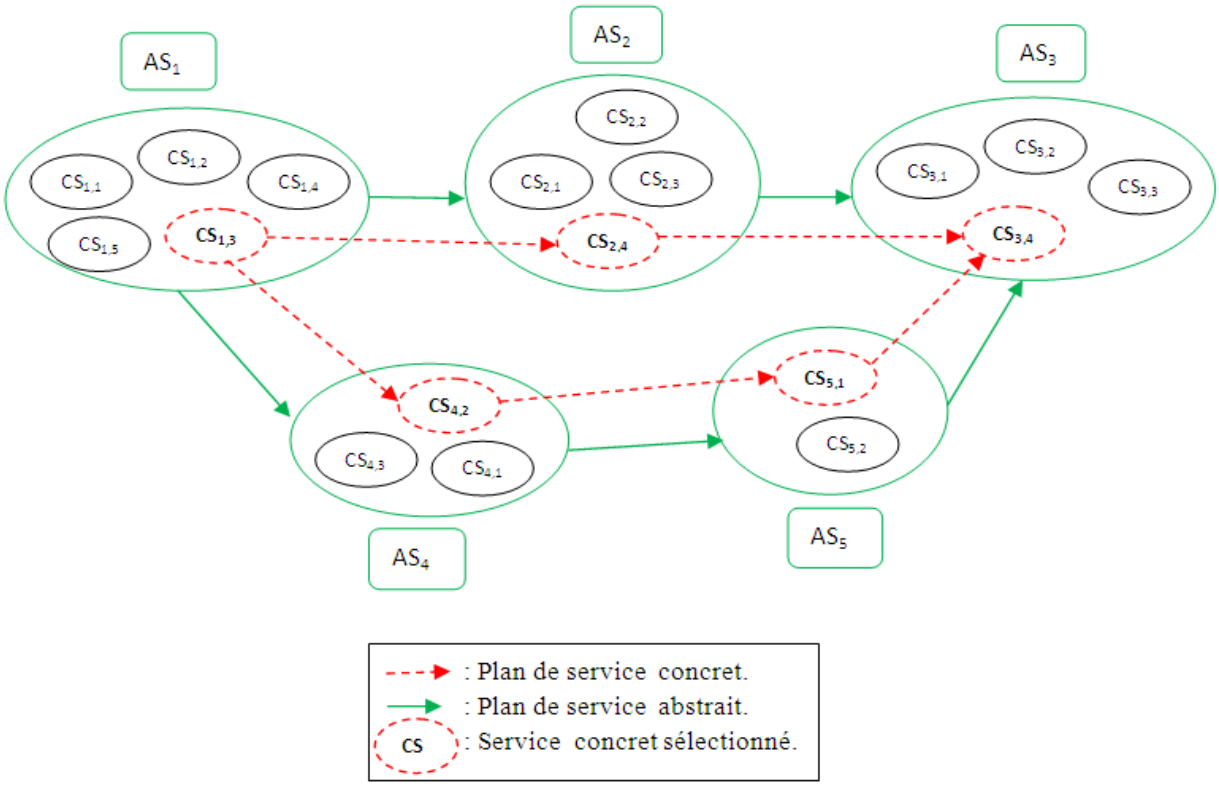


Figure 6.19 : Exemple de sélection d'un service.

Notre modèle de sélection de services se base sur trois phases principales : phase d'estimation de la qualité de service, phase de sélection de services avec apprentissage, et phase d'invocation de services. Tout d'abord, la phase d'estimation de la qualité de service consiste à évaluer la qualité globale d'un service concret en tenant compte de ses paramètres de qualité statique et dynamique. Ensuite, la phase de sélection de services avec apprentissage consiste à choisir réellement le service concret à invoquer. Il s'agit, en faite, de déterminer le meilleur service concret en termes de qualité de service en se basant sur l'estimation fournie par la phase précédente. De plus, la sélection de services utilise un mécanisme d'apprentissage Bayésien afin de tenir compte du caractère incertain des réponses des services concrets. Enfin, la phase d'invocation de services consiste à exécuter réellement le service concret sélectionné dans la phase prétendent en utilisant les valeurs des messages d'entrées de ce service. A la fin de cette exécution, les valeurs des messages de sortie du service exécuté sont reçues et sauvegardées pour une utilisation future. De plus, les valeurs des paramètres de la qualité dynamique de ce même service sont mises à jour grâce à l'observation des différents paramètres de son exécution.

6.4.2. Estimation de la qualité de service

Dans la **section 5.2.2.2 du chapitre 5**, nous avons distingué deux types de paramètres de qualité de service : les paramètres de qualité statique (*SQP : static quality parameters*) et les paramètres de qualité dynamique (*DQP : dynamic quality parameters).* Nous avons souligné que les paramètres de qualité statique restent inchangés pendant une longue période de temps tandis que les paramètres de qualité dynamique changent fréquemment, et ce, en fonction du contexte d'utilisations. Dans cette section, nous proposons deux estimations de la qualité de service fournies respectivement par les paramètres *SQP* et les paramètres *DQP*.

6.4.2.1. Estimation de la qualité de service statique

La qualité de service estimée à partir des paramètres de qualité statique est appelée *qualité de service statique* (*SQoS : Static Quality of Service*). L'estimation de la *SQoS* se base sur une classification des services concrets à l'intérieur de chaque service abstrait. En effet, les services concrets d'un même service abstrait peuvent être classés en fonction de chaque paramètre de qualité statique, puisque ce type de paramètre est fourni initialement et changent rarement. Soit un service abstrait *as* qui possède *N* services concrets. Pour chaque paramètre *SQP* considérée, un ordre de service concret est établie (**Table 6.1**), tels que le service qui offre la meilleure qualité sera en première position (1) et le service qui offre la plus basse qualité sera en dernière position (N). Dans ce classement, la valeur du paramètre de qualité statique SQP_j pour un service concret CS_i est notée $C_{i,j}$ tel que $C_{i,j} \in [0, 1]$. Soit, par exemple, le paramètre de qualité statique « *coût* », alors le service qui a un coût minimum sera classé en première position, et le service qui a le coût maximum sera classé en dernière position.

Services Concrets	SQP_1	SQP_2	……	SQP_k
Service Concret1	$C_{1,1}$	$C_{1,2}$	……	$C_{1,k}$
Service Concret2	$C_{2,1}$	$C_{2,2}$	……	$C_{2,k}$
…	...	...	……	
Service ConcretN	$C_{n,1}$	$C_{n,2}$	……	$C_{n,k}$

Table 6.1 : Classement des services concret par rapport aux paramètres de qualité statique

Les différentes valeurs $C_{i,j}$ sont obtenues en transformant les valeurs initialement fournies pour les différents paramètres de qualité statique à des valeurs comprises entre 0 et 1. Pour ce faire, on calcule d'abord l'écart E_j entre chaque valeur initiale V_j du paramètre de qualité statique SQP_j et son maximum $Max\ (SQP_j)$ selon la formule suivante :

$$E_j = Max(SQP_j) - V_j \quad \ldots (7.1)$$

$Max\ (SQP_j)$ peut être calculé selon l'intervalle des valeurs possibles pour le paramètre SQP_j. Par exemple, pour les valeurs exprimées en pourcentage, le max = 100 %. Tandis que les valeurs exprimées dans un intervalle borné, le max est la borne maximum de cet intervalle. Les paramètres qui prennent des valeurs bien définies (ex. les niveaux de sécurité d'un service), le max est la valeur maximale de ces valeurs définies. Lorsqu'aucune information

n'est disponible sur les bornes max des valeurs, *Max (SQP_j)* prend la valeur maximale de toutes les valeurs disponibles pour le paramètre *SQP_j.* Par exemple, s'il existe N valeurs pour le paramètre *SQP_j* alors *Max (SQP_j)= Max (V_i),* i=1 à N. Une fois tous les écarts *E_j* sont calculés pour le paramètre *SQP_j*, les valeurs des *$C_{i,j}$* peuvent être à présent calculées. Ce calcule se base sur la **formule 7.2** et la **formule 7.3** qui correspondent respectivement au cas de la minimisation du paramètre *SQP_j* et au cas de la maximisation du paramètre *SQP_j*.

$$C_{i,j} = \frac{E_j}{Max(SQP_j)} \dots (7.2) \qquad C_{i,j} = 1 - \frac{E_j}{Max(SQP_j)} \dots (7.3)$$

L'estimation de la qualité statique doit aussi intégrer les préférences des utilisateurs pour chaque paramètre de qualité statique. Ces préférences peuvent être spécifiées en associant un poids *W_i* pour chaque paramètre de qualité statique *SQP_i* afin de refléter son importance pour l'utilisateur. Afin de faciliter à l'utilisateur la spécification de ses préférences, nous avons défini six valeurs pour les poids *W_i* allant de 1 jusqu'à 6, i.e. du moins important au plus important. Nous avons également associé les significations suivantes en terme d'importance pour les valeurs des *W_i*: (*Wi = 1, très faible*), (*Wi = 2, faible*), (*Wi = 3, moyen*), (*Wi = 4, assez élevé*), (*Wi = 5, élevé*), et (*Wi = 6, très élevé*). Dans le cas où aucune spécification ni fournie par l'utilisateur, nous prenons par défaut *Wi = 1*.

En se basant sur tous les paramètres définis précédemment, la qualité de service statique d'un service concret *cs*, notée *SQoS(cs)* est estimée par la **formule 8**. La valeur obtenue pour *SQoS(cs)* reste dans l'intervalle [0, 1].

$$SQoS(cs) = \frac{\sum_k W_k * C_{cs,k}}{\sum_k W_k} \dots (8)$$

Tels que : *$C_{cs,k}$* est la valeur du paramètre *SQP_k* pour le service concret cs, et *W_k* est le poids associé au paramètre *SQP_k.* Dans cette formule, l'indice *k* varie sur l'ensemble des paramètres de qualité statique. Dans ce travail, nous nous limitons à six paramètres de qualité statique à savoir : *Cout monétaire d'un service, Cout de traitement d'un service, Cout de communication d'un service, Niveau de sécurité d'un service, Fiabilité d'un service et Qualité des données produites par un service.* Les trois premiers paramètres sont à minimiser tandis que les trois derniers sont à maximiser lors de la sélection de services.

- **Paramètres de qualité statique à minimiser :**
  - ✓ *Cout monétaire d'un service* : représente les frais générés par l'utilisation d'un service (prix d'une prestation de service).
  - ✓ *Cout de traitement d'un service* : représente la consommation d'un service en termes de ressource de traitement : processeur, mémoire, etc.
  - ✓ *Cout de communication d'un service* : représente le taux de consommation d'un service en termes de ressource de communication (bande passante). Ce paramètre représente le quotient de la bande passante requise par un service sur la bande passante globale.
- **Paramètres de qualité statique à maximiser :**
  - ✓ *Niveau de sécurité d'un service* : est une valeur entre 0 et 3 qui indique le niveau de sécurité assuré par un service en fonction des différents paramètres de sécurité tels

que : la gestion de l'authentification, la gestion des autorisations et le chiffrement des données.

- ✓ *Fiabilité d'un service*: est une valeur entre 0 et 1 qui indique la précision des données transmises par un service.
- ✓ *Performances d'un service*: est une valeur entre 0 et 1 qui indique le degré de performance d'un service (format des données, résolution, etc.).

6.4.2.2. Estimation de la qualité de service dynamique

La qualité de service estimée à partir des paramètres de qualité dynamique est appelée *qualité de service dynamique* (*DQoS : Dynamic Quality of Service*). La *DQoS* estime les fluctuations instantanées que ce soit positives ou négatives que subisse une qualité d'un service. Les fluctuations positives reflètent une amélioration de la qualité de service tandis que les fluctuations négatives reflètent une dégradation de la qualité de service. Ces fluctuations sont dues à la nature dynamique et incertaine de l'environnement ambiant comme nous l'avons souligné dans la **section 1.4.2 du chapitre 1**. Dans cette même section, nous avons mis l'accent sur le faite que la nature dynamique d'un environnement ambiant est due aux changements fréquents qui se produisent alors que sa nature incertaine est due à l'apparition aléatoire de ces changements. D'une façon générale, le changement constant de l'environnement ambiant se traduit par le changement perpétuel du contexte d'utilisation. Plusieurs raisons peuvent être à l'origine de ces changements telles que: l'apparition et la disparition des dispositifs qui hébergent physiquement les services, les défaillances physiques ou défauts d'énergie de ces dispositifs, les déconnexions réseaux, occupation des serveurs, modification ou suppression d'un service, etc. Ces changements de contexte surgissent le plus souvent d'une façon aléatoire et peuvent contribuer soit à l'amélioration ou à la dégradation de la qualité fournie par un service. Ces changements ont également un impact direct sur le comportement des services qui devient pratiquement imprévisible. Nous décrivons ainsi le comportement d'un service par trois paramètres de qualité dynamique à savoir : la *Disponibilité* (*AV : Availability*), *le Temps de réponse* (*RT : Response Time*), *et la Probabilités de réponse* (*PR : Probabilities of Response*).

1. **Disponibilité (*AV : Availability*)**: ce paramètre indique la disponibilité ou l'indisponibilité momentanée d'un service. Ce paramètre prend une valeur booléenne 0 ou 1. En effet, à un instant donné, un service peut être disponible (*i.e. AV = 1*) ou indisponible (*i.e. AV = 0*). La disponibilité est un paramètre imprévisible car il dépend de plusieurs facteurs notamment le niveau d'énergie de la ressource de traitement, la mobilité de la ressource, la disponibilité des serveurs, l'état de la connexion réseau, etc. L'indisponibilité d'un service peut se produire dans deux cas de figure: soit avant ou pendant l'invocation du service. Par exemple, la mobilité des dispositifs ou bien le retrait d'un service par le fournisseur de service provoquent l'indisponibilité du service avant même son invocation. Alors que certaines pannes qui surviennent pendant l'exécution d'un service telles que les déconnexions réseaux, le crash d'un serveur, et la panne du dispositif hébergeant un service peuvent provoquer une indisponibilité momentanée du service invoqué et donc un échec de l'invocation du service.
2. **Temps de réponse (*RT : Response Time*)**: le temps de réponse, appelé aussi latence, représente le temps requis par un service pour répondre à une demande (invocation). En

d'autres termes, le temps de réponse d'un service est la durée qui sépare le début de son invocation ($T_{DebutInvocation}$) et l'instant de la réception ($T_{FinInovcation}$) de tous les messages de sortie demandés à ce service. Ainsi, le temps de réponse d'un service concret *cs*, noté *RT (cs)*, se calcule par la formule suivante :

$$RT(cs) = T_{FinInovcation}(cs) - T_{DebutInvocation}(cs) \dots (9.1)$$

Le temps de réponse d'un service est aussi un paramètre imprévisible car il dépend de plusieurs facteurs notamment l'état de la connexion réseau, l'occupation des ressources de calcul, la charge des serveurs, etc. Afin de mieux contrôler l'invocation des services concrets, nous introduisons la notion de *délai d'attente* ou simplement *timeout*. Ce dernier représente le délai maximum d'attente de la réponse d'un service invoqué. Au delà de ce *timeout*, si aucune réponse n'est reçue du service invoqué, ce service sera considéré comme étant indisponible. Toutefois, la spécification du *timeout* doit tenir compte du temps de réponse de chaque service. En effet, l'inconvénient majeur d'avoir un seul *timeout* pour tous les services c'est la perte considérable en termes de délai d'attente des services. Etant donné que le *timeout* spécifié doit être supérieur au temps de réponse le plus long, les services dont le temps de réponse est court consommeront inutilement des *timeout* assez élevés. Ainsi, c'est tout le système qui sera pénalisé car il sera cadencé par des délais d'attente très longs. Afin de palier à cette contrainte, il est plus judicieux d'avoir un *timeout* qui soit propre à chaque service, et ce, en fonction de son temps de réponse. Dans ce cas de figure, l'invocation d'un service concret *cs* dont le temps de réponse est *RT (cs)* sera contrôlée par un *timeout* spécifié par la formule suivante :

$$timeout(cs) = \alpha . RT(cs) \text{ avec } \alpha > 1 \dots (9.2)$$

3. **Probabilité de réponse (*PR : Probabilities of Response*)** : ce paramètre reflète le comportement imprévisible d'un service qui peut basculer d'un état à un autre d'une façon aléatoire. En d'autres termes, la probabilité de réponse d'un service est la probabilité que, lors de son invocation, ce service répond par l'ensemble des messages de sortie requis dans le délai spécifié (*timeout*). La probabilité de réponse dépend de la qualité de la connexion réseau et la disponibilité des ressources de traitement (dispositifs). Elle dépend également de la fréquence des changements (pannes, déconnexions, indisponibilité momentanée des serveurs, suppression ou modification d'un service, etc.) qui surviennent dans l'environnement ambiant. Nous considérons un service concret comme étant une action à exécuter dans un environnement ambiant. Etant donné que cet environnement est incertain, l'exécution d'une action peut mener à deux états imprévisibles : un *état positif* avec une probabilité *p* et un *état négatif* avec une probabilité *q=1-p*. Un état d'un service représente la réponse qu'il fournie après son exécution (invocation). Deux types de réponse sont alors distingués : réponse positive et réponse négative.
   - **Réponse positive** : la réponse d'un service invoqué est considérée comme étant *positive* si ce service répond par l'ensemble des messages de sortie requis dans le délai (*timeout*) qui lui est spécifié. Cela signifie que, après l'exécution du service, toutes les valeurs seront affectées aux paramètres de ses messages de sortie. Ces valeurs représentent, en quelque sorte, l'effet produit par le service exécuté. Cet état (i.e.

positif) est souhaitable puisque nous pouvons, à présent, utiliser directement les valeurs obtenues de l'exécution du service ou bien les utiliser pour invoquer d'autres services.

- **Réponse négative** : soit le service invoqué ne répond pas. Plusieurs raisons peuvent être à l'origine de cet état telles que : la panne du réseau, la maintenance du serveur, la suppression du service,…etc. Dans ce cas, l'état du service est inconnu. Ou bien le service invoqué ne fourni pas les fonctionnalités demandées. Il se peut que le service ait été modifié ou bien il a changé de contexte. Dans cet état, nous trouvons également les services qui tombent en panne lors de leur exécution. Cet état (i.e. *négatif*) regroupe tous les différents échecs que nous avons déjà discutés. Aussi, quand la raison est inconnue, la réponse est considérée comme *négative*.

Dans notre modèle, les probabilités représentent l'estimation de la réponse de chaque action (service). Au départ, il est possible d'avoir des estimations de ces probabilités selon certaines connaissances sur les services tels que la disponibilité et la fiabilité. Toutefois, ces connaissances initiales peuvent être inexactes ou complètement inconnus. Par conséquent, l'introduction des mécanismes d'apprentissage devient nécessaire. Il est possible d'introduire, à ce niveau, plusieurs modèles d'apprentissage. Le modèle d'apprentissage Bayésien semble bien adapté à notre cas.

Apprentissage Bayésien: Lorsqu'on ignore les lois qui régissent un environnement, l'évolution des probabilités des différentes actions à exécuter dans cet environnement reste inconnue. Dans ce cas, le modèle d'apprentissage Bayésien permet d'apprendre petit à petit les probabilités de ces actions à travers les interactions avec l'environnement. Dans ce modèle d'apprentissage, quand les estimations initiales sont complètement inconnues, des probabilités égales sont attribuées pour chaque action. Par la suite, les réponses obtenues à travers l'exécution des actions sont utilisées pour réviser les estimations initiales. De cette manière, l'exécution des actions et l'apprentissage sont intercalés. L'algorithme d'apprentissage Bayésien **[302]** maintien un conteur d'expériences, noté *Experience,* pour chaque action à exécuter. Ce compteur est initialisé à 1 et indique le nombre de fois qu'une action a été exécutée. Pour illustrer la manière de mettre à jour des probabilités des actions par l'apprentissage Bayésien, nous considérons le cas suivant : soit une action *a* à exécuter dans un environnement incertain afin de modifier l'état d'un système qui était initialement dans l'état *s*. L'exécution de l'action *a* fait passer le système à l'un des deux états suivants: un état *s* (le système reste dans le même état) avec une probabilité *p* et un état s' (le système passe à un nouvel état) avec une probabilité *q*. On peut noter ces deux probabilités comme suit : $p=T(s,a,s)$: probabilité de transition de l'état *s* à l'état *s* en exécutant l'action *a* et $q=T(s,a,s')$: probabilité de transition de l'état *s* à l'état *s'* en exécutant l'action *a*. Etant donné qu'il n ya que deux états possibles *s* et *s',* donc $T(s,a,s)+ T(s,a,s')=1$. Supposant qu'après l'exécution de l'action *a*, le système passe à l'état s', le modèle d'apprentissage Bayésien modifie alors les probabilités de passage *T* en probabilités *T'* de la manière suivante :

$$\mathrm{T}'(\mathrm{s},\mathrm{a},\mathrm{s}') = \frac{\big(\mathrm{T}(\mathrm{s},\mathrm{a},\mathrm{s}') \times Experience(a)\big) + 1}{Experience(a) + 1} \dots (10.1)$$

$$\mathrm{T}'(\mathrm{s},\mathrm{a},\mathrm{s}) = \frac{\big(\mathrm{T}(\mathrm{s},\mathrm{a},\mathrm{s}) \times Experience(a)\big)}{Experince(a) + 1} \dots (10.2)$$

A travers cette modification, nous constatons que la nouvelle probabilité $\mathrm{T}'(\mathrm{s},\mathrm{a},\mathrm{s}')$ augmente sensiblement par rapport à l'ancienne probabilité $\mathrm{T}(\mathrm{s},\mathrm{a},\mathrm{s}')$ tandis que la nouvelle probabilité $\mathrm{T}'(\mathrm{s},\mathrm{a},\mathrm{s})$ diminue sensiblement par rapport à l'ancienne probabilité $\mathrm{T}(\mathrm{s},\mathrm{a},\mathrm{s})$. Par ailleurs, nous pouvons facilement vérifier que la somme des nouvelles probabilités obtenues est toujours maintenue à 1, i.e. $\mathrm{T}'(\mathrm{s},\mathrm{a},\mathrm{s}') + \mathrm{T}'(\mathrm{s},\mathrm{a},\mathrm{s}) = 1$. A la fin de cette mise à jour, le compteur d'expérience associée à l'action exécutée, i.e. *Experience (a)* est incrémentée de 1 pour les futures mises à jour.

Par analogie au cas précédent, l'apprentissage Bayésien peut s'appliquer sur un service concret puisque ce dernier est une action qui peut mener à deux états différents après son invocation: un *état positif* avec une probabilité *p* et un *état négatif* avec une probabilité *q=1-p.* Etant donnée que la probabilité de réponse (*PR*) d'un service concret *cs* est sa probabilité de donner une réponse (état) positive après son invocation, alors l'apprentissage Bayésien sur *PR(cs)* peut s'appliquer de la manière suivante :

$$PR_{new}(\mathrm{cs}) = \frac{\big(PR_{old}(cs) \times Experience(cs)\big) + AV(cs)_{new}}{Experience(cs) + 1} \dots (10.3)$$

Avec : $PR_{old}(cs)$ est l'ancienne probabilité de réponse de *cs*, $PR_{new}(\mathrm{cs})$ est la nouvelle probabilité de réponse de *cs*, et $AV(cs)_{new}$ est la nouvelle valeur de la *disponibilité* de *cs* tels que : $AV(cs)_{new} = 1$ si la réponse de *cs* est positive et $AV(cs)_{new} = 0$ dans le cas contraire.

Finalement, en se basant sur les trois paramètres de qualité dynamique décrits précédemment à savoir : *la disponibilité, le temps de réponse* et *la probabilité de réponse*, nous pouvons estimer la qualité dynamique *DQoS* fournie par un service concret. Il est clair que la qualité dynamique d'un service vise la minimisation du temps de réponse et la maximisation de la probabilité de réponse. Ainsi, la qualité dynamique d'un service concret *cs* se calcule à l'aide de la formule suivante :

$$DQoS(\mathrm{cs}) = \frac{PR(cs)}{RT(cs) + 1} \dots (11)$$

Etant donné que le paramètre *Disponibilité* est déjà utilisé lors du calcul de la probabilité de réponse (**Formule 10.3**), donc ce paramètre est implicitement utilisé pour le calcul de la *DQoS*. La valeur 1 est ajoutée au dénominateur dans la **formule 11** afin d'obtenir une valeur de *DQoS(cs)* dans l'intervalle [0, 1], i.e. même intervalle de variation que la *SQoS(cs)*.

6.4.2.3. Estimation de la qualité de service globale

La qualité de service global est notée simplement *qualité de service* (*QoS : Quality of Service*). La qualité de service globale doit tenir compte à la fois de la qualité statique et de la qualité dynamique fournies par chaque service concret. Pour cela, la qualité globale attendue d'un service concret à l'instant *t* doit tenir compte de la qualité dynamique attendue à l'instant *t* et de la qualité statique qui reste constante. Donc, intuitivement, la qualité global d'un service concret *cs* à l'instant *t* peut être estimée suivant la formule suivante:

$$\mathrm{QoS_t}(\mathrm{cs}) = \mathrm{SQoS}(\mathrm{cs}) \times \mathrm{DQoS}(\mathrm{cs})_{\mathrm{t}} \dots (12.1)$$

Dans cette formule, les fluctuations sur la *QoS* sont dues uniquement aux fluctuations engendrées par la *DQoS* puisque la *SQoS* reste constante. En effet, la *SQoS* joue le rôle d'un facteur de pondération de la *DQoS*. Par conséquent, la **formule 12** tient compte uniquement des changements qui affectent localement un service concret *cs* et ne prend pas en considération les changements qui affectent globalement l'environnement ambiant. Ces changements sont caractérisés notamment par la fréquence de départ et d'arrivée des services dans l'environnement ambiant. Le départ (disparition) d'un service peut survenir dans les deux cas suivants : soit le retrait momentané ou définitif du service du répertoire de services par son fournisseur ou bien le passage de l'état disponible à l'état indisponible après son invocation. Par ailleurs, l'arrivée (apparition) d'un service peut survenir dans les deux cas suivants : soit le service est nouvellement ajouté au répertoire de services par son fournisseur ou bien le passage de l'état indisponible à l'état disponible après son invocation. Les arrivées et les départs des services abstraits peuvent être notifiés soit par le module de découverte et de classification de services soit par le module de monitoring de services (**section 6.5**). Ainsi, si on pose $SA_{[T]}$ le nombre de services apparus dans un intervalle de temps T et $SD_{[T]}$ le nombre de services disparus dans ce même intervalle de temps, alors la variation globale de l'environnement ambiant dans l'intervalle de temps T, notée $VE_{[T]}$ peut être estimée par la formule suivante :

$$VE_{[T]} = SA_{[T]} + SD_{[T]} \dots (12.2)$$

On constate que, dans un environnement hautement dynamique, le nombre de services apparus et disparus augmente, donc la variation de l'environnement augmente. Contrairement à un environnement moins dynamique ou relativement stable, le nombre de services apparus et disparus diminue, donc la variation de l'environnement diminue aussi. Il en résulte que, l'importance qu'on accorde à la qualité dynamique et la qualité statique dépend du degré de variation de l'environnement. Autrement dit, si on a faire à un environnement relativement stable ou constant, on accorde plus d'importance à la qualité statique car dans ce cas de figure les services sont, tous ou presque, disponibles ($AV=1$) et leurs probabilité de réponse et quasi certaine ($PR \cong 1$). Donc il n'est pas nécessaire d'accorder trop d'importance à la qualité dynamique. Dans le cas contraire, i.e. environnement très dynamique, l'importance sera accordée à la qualité dynamique car la disponibilité des services est variable et leurs réponse n'est pas certaine. Donc, il est nécessaire de cerner le service qui fournie une qualité dynamique bien meilleure afin de garantir sa réponse. Ainsi, la qualité globale d'un service peut être estimée en associant des pondérations différentes pour la qualité statique et la qualité dynamique selon la formule suivante :

$$\mathrm{QoS(cs)_t} = \frac{\alpha \mathrm{SQoS(cs)} + \beta \mathrm{DQoS(cs)_t}}{\alpha + \beta} \dots (12.3)$$

De telle sorte que si *VE* augmente, i.e. l'environnement est dynamique, on donne plus d'importance à β. Dans le cas contraire, l'environnement est relativement constant, donc on donne plus d'importance à α. En d'autres termes, $\beta = f_1(VE)\ et\ \alpha = f_2(VE)$ avec $f_1$est une fonction croissante et f_2 est une fonction décroissante. A titre d'illustration, si on limite le

nombre de variation maximum de l'environnement à une constante C, on peut définir, par exemple, les deux fonctions simples suivantes : $\beta = VE$ et $\alpha = C - VE$.

6.4.3. Invocation des services concrets

La procédure complète d'invocation d'un service concret est détaillée par l'algorithme *CSI* (*Concrete Service Invocation*) de la **figure 6.20**. Comme le montre cet algorithme, l'invocation d'un service concret comprend trois phases essentielles : préparation de l'exécution du service, exécution du service, et mise à jour des informations sur le service. Tout d'abord, dans la phase de préparation de l'exécution du service, i.e. **ligne 1 et 2**, la procédure d'invocation construit les messages d'entrées du service à exécuter à partir des valeurs de contexte contenues dans une table globale appelée *Table of Satisfied Messages (TSM).* Cette table est crée afin de maintenir tous les messages produits après l'invocation des services concrets. Concrètement, la table *TSM* conserve toutes les données de contexte produites dans les messages de sortie en utilisant l'approche clé-valeur (*key-value*). Par exemple, un message *M* qui a la structure suivante: *M <temperature, humidity>,* et il est produit par un service concret comme *M <30, 70>,* sera sauvegardé dans le table *TSM* comme suit: *<temperature, 30>, <humidity, 70>*. Pour construire le message *M* à partir de la table *TSM*, il suffit de chercher l'existence des deux clés *temperature* et *humidity* comme entrées dans la table *TSM*, puis récupérer leurs valeurs respectives.

Concrete Service Invocation (CSI)

Inputs: Concrete Service: CS, Required Messages: RM

Output: Concrete Service Response: CSR

Begin

1. CSR=False
2. Get CS.inputs from Table of Satisfied Messages TSM
3. Call CS with its inputs
4. Get CS.timeout using formula 9.2
5. **While** (CS.timeout is not expired and RM are not received) **do**
6. Wait and Receive outputs from CS
7. **End while**
8. Update CS.RT
9. Update CS.timeout using formula 9.2
10. **If** (RM are received) **then**
11. CSR=True
12. CS.AV=1
13. TSM.Add (RM)
14. **Else**
15. CS.AV=0
16. **End if**
17. Increment CS.experience
18. Update CS.PR using formula 10.3

End

Figure 6.20 : Algorithme d'invocation d'un service concret

Ensuite, la phase exécution du service, i.e. **ligne 3** jusqu'à la **ligne 7**, consiste à exécuter réellement le service en l'appelant par la valeur de ses messages d'entrées construits dans la phase précédente. Après le lancement de l'exécution du service (**ligne 3**), l'algorithme *CSI* attend jusqu'à la réception de tous les messages requis (*RM : Required Messages*) comme sortie du service exécuté, et ce, dans un délai d'attente bien déterminé qui est le *timeout* de ce service. Enfin, la phase de mise à jour des informations sur le service consiste à mettre d'abord à jour le temps de réponse (*RT*) du service exécuté (**ligne 8**), puis son *timeout* (**ligne 9**) puisque ce dernier dépend du temps de réponse. En effet, le temps de réponse réel d'un service c'est le temps pris par ce service pour fournir tous les messages requis. Dans le cas ou tous ces messages requis sont reçus alors la réponse du service exécuté (*CSR : Concrete Service Response*) est considérée comme positive (*True*) et ce service sera considéré comme disponible (*AV=1*). De plus, les valeurs obtenues des messages de sorties du service exécuté sont ajoutées à la table *TSM*. Dans le cas contraire, i.e. les messages requis ne sont pas tous reçus, alors le service exécuté sera considéré comme indisponible (*AV=0*). Enfin, dans tous les cas de figure, le nombre d'invocation du service exécuté (*Expérience*) sera incrémenté et sa probabilité de réponse sera mise à jour par l'apprentissage Bayésien. Comme résultat final, l'algorithme *CSI* retourne la valeur de la réponse *CSR* obtenue par l'invocation du service.

6.4.4. Sélection des services concrets avec apprentissage

La procédure complète de sélection d'un service concret est détaillée par l'algorithme *CSS* (*Concrete Service Selection*) de la **figure 6.21**. La procédure de sélection de service consiste à choisir un service concret parmi un ensemble de services concrets contenus dans un service abstrait donné. Notre méthode de sélection prend en compte la qualité de service globale estimée pour chaque service concret. Comme nous l'avons décrit dans la **section 4.2.3**, la qualité globale d'un service concret à l'instant *t* prend en compte la qualité dynamique à l'instant *t* de ce service ainsi que sa qualité statique. Cette qualité globale peut être estimée en utilisant soit la **formule 12.1** ou bien la **formule 12.3**. Par conséquent, le service concret *cs* qui sera invoqué à l'instant *t* est choisi parmi les services concret d'un service abstrait *AS* selon la formule suivante:

$$SelectedConcretSevice_t(\mathrm{AS}) = \mathrm{ArgMax}\big(\mathrm{QoS_t(cs)}\big)_{\mathrm{cs}\in\mathrm{AS}} \dots (13)$$

Toutefois, avant d'appliquer la **formule 13** pour choisir un service concret, deux pré-conditions doivent être vérifiées. Tout d'abord, il s'agit de vérifier l'existence de tous les messages d'entrée du service abstrait *AS* dans la table *TSM* (**section 4.3**). Par exemple, pour savoir si un message *M <temperature, humidity>* est satisfait dans la table *TSM*, il suffit de chercher l'existence des deux clés *temperature* et *humidity* comme entrées dans la table *TSM*. Ensuite, lorsque la catégorie du service abstrait *AS* est un service d'acquisition de contexte, i.e. sa catégorie est *context acquisition*, alors l'algorithme *CSS* vérifie également la localisation de chaque service concret appartenant à *AS*. Cette localisation doit correspondre à celle de l'événement détecté (i.e. *Event Location EL*). En effet, le contexte doit être acquis à partir de l'emplacement ou l'événement s'est produit. Ces deux pré-conditions sont vérifiées par la procédure *checkPreconditions* à la **ligne 2**. Après cette vérification, un sous ensemble de services concrets qui satisfassent ces pré-conditions appelé *satisfiedServices* est construit.

Dans le cas ou cet sous ensemble n'est pas vide, la qualité dynamique de tous ses services concrets sera évaluée. Afin de calculer cette qualité dynamique dans la phase initiale (*i.e.* $t = 0$), nous supposons que tous les services concrets sont équiprobables en terme de *réponse positive*, i.e. PR_0 est égal à 0.5 pour l'ensemble des services concrets. De plus, tous les services concrets sont disponibles ($AV = 1$) et leurs temps de réponse initial RT_0 est aussi connu. Par la suite, ces paramètres de qualité dynamique sont mis à jour après chaque invocation d'un service concret. De plus, nous supposons que la qualité statique *SQoS* est déjà calculée pour l'ensemble des services concrets lors de la phase de découverte et de classification de services. Ainsi, le meilleur service concret en terme de qualité globale fournie est sélectionné en appliquant la **formule 13** sur le sous ensemble de services satisfaits *satisfiedServices (***ligne 8**). Par la suite, le service concret sélectionné est invoqué à la **ligne 9**, en utilisant la procédure *CSI* décrit précédemment. A chaque fois que cette procédure est appelée, les paramètres de qualité dynamiques correspondant au service concret invoqué sont mis à jour. Donc, la qualité dynamique *DQoS* du service concret invoqué doit être également mise à jour à **la ligne 10**.

Concrete Service Selection (CSS)

Inputs: Abstract Service: AS**,** Required Messages: RM, Event Location: EL Learning Threshold: LT

Output: Abstract Service Response: ASR

Begin

1. ASR=False
2. satisfiedServices=checkPreconditions (AS, EL)
3. **If** (satisfiedServices≠∅)
4. **For each** Concrete Service CS in satisfiedServices **do**
5. Compute CS.DQoS using formula 11
6. Compute CS.QoS using formula 12.1 or 12.3
7. **End for**
8. **Repeat**
9. Select Concrete Service CS from satisfiedServices using formula 13
10. ASR=CSI (CS, RM)
11. Update CS.DQoS using formula 3
12. **Until** (ASR=True) or ($\mu(CS.P_{AI}, CS.P_{BI}) < LT$)
13. End If

End

Figure 6.21 : Algorithme de sélection d'un service concret

Une réponse appelée *ASR (Abstract Service Response*) est associée au service abstrait *AS* qui est l'objet de la procédure de sélection. *ASR* sert à récupérer la réponse retournée par la procédure *CSI*. Pour chaque service concret *CS* invoqué, l'algorithme *CSS* maintient sa probabilité de réponse avant l'invocation (P_{BI}*: Probability Before Invocation)* et sa probabilité de réponse après l'invocation (P_{AI}*: Probability After Invocation).* Ces deux

probabilités sont utilisées afin de calculer une distance de probabilité appelée μ. Par exemple, cette distance peut être calculée en utilisant la distance euclidienne suivante :

$$\mu(P_{AI}, P_{BI}) = |P_{AI} - P_{BI}| \dots (14)$$

Lorsque la réponse obtenue après l'invocation d'un service concret est *positive* (i.e. *ASR = True*), la sélection se termine. Sinon, la procédure de sélection effectue une nouvelle sélection afin de trouver un autre service concret. Ainsi, la sélection est renouvelée tant que la réponse est toujours *négative* (i.e. *ASR = False*) et la distance de probabilité $\mu(CS.P_{AI}, CS.P_{BI})$ est toujours supérieur au seuil d'apprentissage spécifié (*learning threshold LT*). En effet, l'algorithme *CSS* augmente la sélection de services par un apprentissage Bayésien. Une fois le seuil d'apprentissage spécifié est atteint sans trouver aucun service concret approprié, la réponse du service abstrait *AS* est considérée comme *négative* (*ASR = false*). Comme résultat final, l'algorithme *CSS* retourne la valeur de la réponse *ASR* obtenue après la sélection et l'apprentissage.

6.5. Modèle de monitoring de services

6.5.1. Principe du monitoring de services

Le processus de monitoring des services consiste à contrôler trois aspects principaux à savoir: la composition de services, la sélection de services et l'invocation de services. Ce processus définit une coordination appropriée entre ces trois aspects afin d'atteindre concrètement l'objectif désiré quand un certain événement survienne dans un environnement ambiant. Cette coordination est réalisée suivant une nouvelle stratégie flexible, tolérante aux pannes et tient compte du contexte courant d'utilisation. Autrement dit, l'objectif de la stratégie de monitoring de services consiste à augmenter les chances d'exécution d'un graphe de composition de services dans un environnement dynamique et incertain toute en garantissant une meilleure qualité de service. Il s'agit notamment de tenir compte des différentes pannes des services qui peuvent survenir d'une façon imprévisible. Dans de telles conditions, le mécanisme de monitoring doit assurer une continuité de service même parfois en mode dégradé, i.e. avec une qualité abaissée. Afin d'atteindre ces objectifs, notre stratégie se base sur deux mécanismes de substitution de services : un mécanisme de substitution local et un mécanisme de substitution global. Le mécanisme de substitution local consiste soit à remplacer un service concret par un autre service concret dans le même service abstrait, ou bien à remplacer un service abstrait par un autre service abstrait fournissant certaines fonctionnalités similaires. Le mécanisme de substitution global consiste, quant à lui, à effectuer une recomposition de tous les services disponibles. Ces différents mécanismes de substitutions sont intégrés dans une approche en couches (**Figure 6.22**) afin de séparer trois niveaux principaux de monitoring de services à savoir: le niveau initial de monitoring de services, le niveau supérieur de monitoring de services et le niveau inférieur de monitoring de services. A chaque niveau, un ou plusieurs algorithmes sont intégrés pour effectuer une tâche spécifique. Ces algorithmes sont essentiellement fournis par les modules de composition, de sélection et d'invocation de services.

La stratégie globale de monitoring de services est illustrée sur le schéma de la **figure 6.22.** Cette stratégie implémente concrètement le module *Services Monitoring* du Framework *FrEvASeC* (**section 5.3 du chapitre 5**). Ce module constitue le noyau central du Framework *FrEvASeC* car il coordonne les autres modules afin d'atteindre des objectifs bien précis, spécifiés dans des règles événementielles (*events rules*). Pour les besoins de son fonctionnement, le module de monitoring possède un accès direct à trois types de base de données : la base des services abstraits (*ASDB*), la base des sous graphes (*SGDB*) et la base des spécifications des utilisateurs (*USDB*). Le module de monitoring possède également un accès indirect à deux autres types de base de données : la base du graphe global (*GGDB*) et la base des services concrets (*CSDB*). L'accès à ces deux dernière bases est indirect car il est effectuer via respectivement les modules de composition de services et les modules de sélection et d'invocation de services comme le montre la **figure 6.22**. Pour rappel, *GGDB* contient le graphe global de tous les services disponibles tandis que *SGDB* contient les sous graphes associés à certains événements déjà détectés. Le module de monitoring coordonne également son fonctionnement avec deux autres modules à savoir: le module de détection d'événements et le module de notification et de contrôle de dispositifs. En effet, le module de monitoring reçoit d'abord les événements détectés du module de détection d'événements, ensuite, il envoie le résultat du monitoring de ces événements au module de notification et de contrôle de dispositifs afin de notifier l'utilisateur concerné.

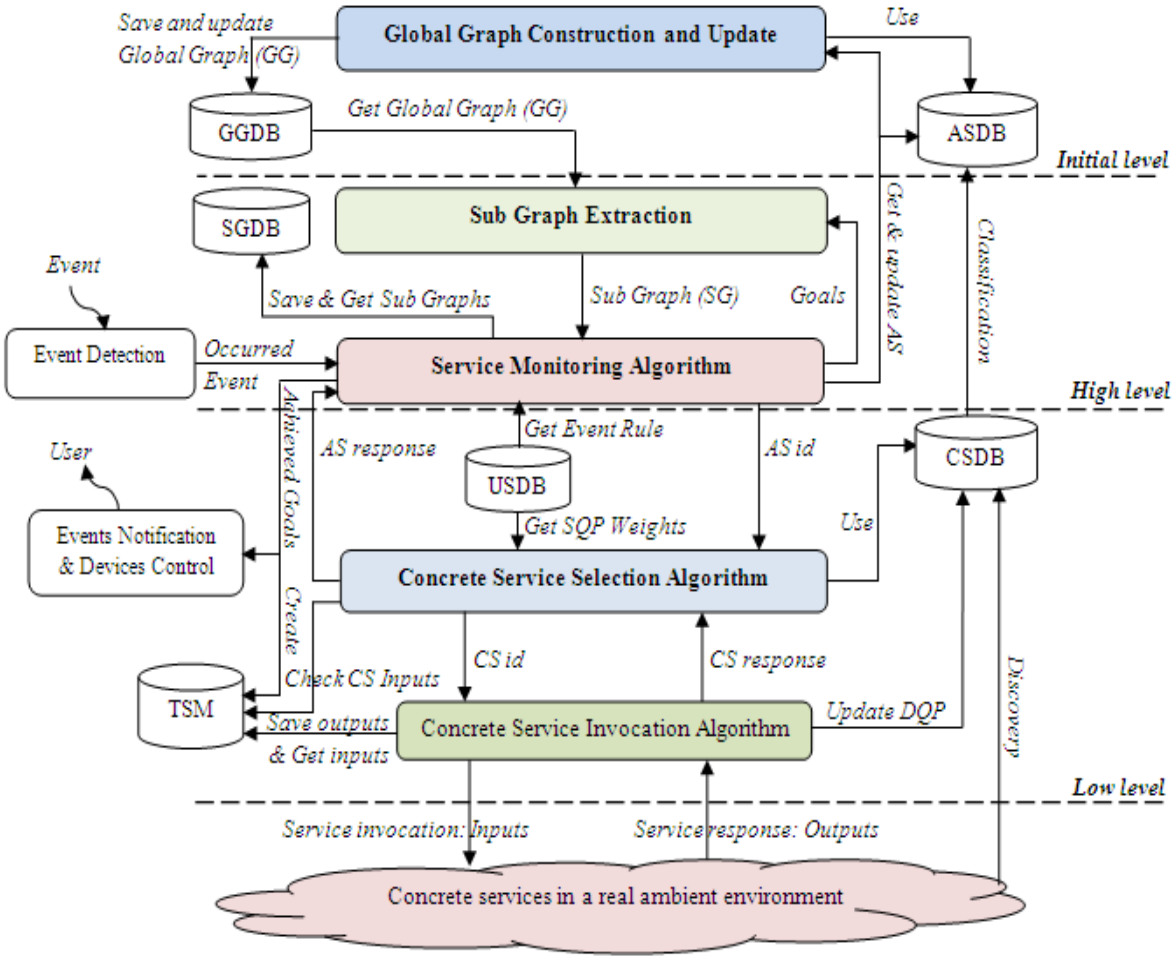


Figure 6.22 : Architecture de la stratégie de monitoring de services

6.5.2. Niveau initial de monitoring de service

Au niveau initial de monitoring, un graphe global (*GG*) de tous les services disponibles dans la base de données des services abstraits (*ASDB*) est créée en reliant, d'une manière appropriée, les entrées et sorties de ces services en utilisant l'algorithme de construction d'un graphe global (*AGoSC construction algorithm*) décrit dans la **section 6.3.4.1**. Ainsi, le graphe *GG*, équivalent au graphe *AGoSC,* est construit en *offline* afin de réduire le temps de réponse à des éventuels événements qui surviennent dans l'environnement ambiant. De plus, ce graphe est construit d'une manière abstraite, c'est à dire sur un ensemble de services abstraits afin de pouvoir s'adapter facilement aux changements qui peuvent se produire sur les services concrets et le contexte de l'environnement. Le graph global obtenu est sauvegardé dans la base de données du graphe global (*GGDB*) pour une utilisation ultérieure par les autres niveaux de monitoring de services. Par ailleurs, à ce niveau initial, la table globale des messages satisfaits *TSM*, décrite dans la **section 6.4.3**, est crée et initialisée afin de maintenir tous les messages produits par les services concrets après leur invocation. Comme nous l'avons déjà mentionné dans la **section 6.4.3**, la table *TSM* conserve toutes les données de contexte produites dans chaque message de sortie en utilisant l'approche clé-valeur (*key-value*). Pour savoir, par exemple, si un message est satisfait dans la table *TSM*, il suffit de chercher l'existence de toutes les clés de ce message comme entrées dans la table *TSM*. La procédure complète d'initialisation exécutée à ce niveau du monitoring est détaillée par l'algorithme *InitA* (*Initialization Algorithm*) de la **figure 6.23**.

Initialization Algorithm (InitA)

Inputs: Abstract Services Data Base: ASDB

Begin

1. Create and initialize Table of Satisfied Messages TSM
2. GlobalGraph=GlobalGrapheConstruction()
3. Save GlobalGraph in Graph Data Base

End

Figure 6.23 : Algorithme d'initialisation

Le graphe global *GG* est mis à jour dans deux cas de figures: soit lors du départ d'un ou plusieurs services abstraits ou bien lors de l'arrivée d'un ou plusieurs services abstraits. Le départ d'un service est considéré comme une suppression d'un service du graphe global tandis que l'arrivée d'un service est considérée comme un ajout d'un service au graphe global. Quand un ou plusieurs services abstraits sont supprimés du graphe *GG*, ce dernier est mis à jour en supprimant d'abord tout les successeurs des services supprimés, puis en effectuant une nouvelle reconstruction du graphe *GG* juste à partir du premier niveau où au moins service abstrait est supprimé. En effet, la suppression d'un service du graph *GG* affecte uniquement les successeurs du service supprimé. Donc la mise à jour touche seulement les niveaux supérieurs au niveau du service supprimé dans le graphe *GG*. Pareil lors de l'ajout des services abstraits au graphe *GG*. La mise à jour de ce graphe commence à partir du premier niveau dans lequel au moins un service abstrait est ajouté.

6.5.3. Niveau supérieur de monitoring de service

Lorsqu'un événement se produit dans l'environnement ambiant, il sera détecté et signalé par le module de détection d'événement du Framework *FrEvASeC*. Ce module transmet l'événement détecté au niveau supérieur de monitoring de services. Plus précisément, l'événement sera transmis à l'algorithme de monitoring de services *SeMA* (*Services Monitoring Algorithm*) décrit dans la **figure 6.24**. L'algorithme *SeMA* vérifie d'abord dans la base de données des sous graphes (*SGDB*) l'existence d'un sous-graphe pour l'événement détecté. En effet, un sous-graphe peut être sauvegardé grâce aux monitorings des événements précédents. Dans le cas ou aucun sous graphe n'est disponible pour l'événement détecté (i.e. *SubGraph.size = 0*), *SeMA* extrait alors un sous-graphe (*SG*) du graphe global (*GG*) construit dans le niveau initial de monitoring. Cette extraction se base sur l'algorithme d'extraction d'un sous graphe (*SubGoSC Extraction Algorithm*) décrit dans la **section 6.3.4.2**. Le sous graphe *SG* extrait doit satisfaire la règle événementielle (*ER*) associée à l'événement détecté. Cette règle est extraite de la base des spécifications des utilisateurs (*USDB*). Par la suite, le sous graphe extrait est transmis pour exécution dans un ordre séquentiel des niveaux en utilisant l'algorithme de sélection de services concrets *CSS* décrit dans la **section 6.4.4**. Lorsque la réponse d'un service abstrait est *négatif* (**ligne 17**), ce service sera retiré du sous-graphe *SG* et du graphe global *GG*. En outre, ce service sera considéré comme indisponible dans la base *ASDB* et un processus de mise à jour du graphe *GG* sera lancé à la **ligne 20**. Afin de remplacer le service défectueux, l'algorithme *SeMA* utilise, dans un premier temps, un mécanisme de substitution local. En effet, *SeMA* tente de remplacer le service tombé en panne par un nouveau service abstrait AS_i au même niveau du graphe *SG* où cette panne s'est produite (**ligne 22**). *SeMA* effectue alors une sorte de recherche spécifique de services dans la base *ASDB*. Cette recherche se base sur la condition de remplacement spécifiée à la **ligne 21**. Cette condition stipule que les paramètres contenus dans tous les messages d'entrée du nouveau service AS_i doivent être inclus dans la table *TSM*, et les messages de sorties de ce même service doivent inclurent les messages de sortie attendus appelés *Required Messages* (*RM*). Lorsqu'aucun service abstrait n'est trouvé pour remplacer le service défectueux, *SeMA* utilise alors un mécanisme de substitution global qui consiste à faire une recomposition (re-planification) de services. Afin de contrôler ce processus de recomposition, un seuil de recomposition appelé *Replanning Threshold* est initialement fixé. Lorsque le nombre de recomposition effectué par *SeMA* n'a pas encore atteint ce seuil de recomposition fixé, une nouvelle extraction d'un sous graphe de services peut être alors déclenchée **à la ligne 27**. Cette extraction est effectuée sur le graphe global *GG* déjà mis à jour à la **ligne 20**. Le processus de monitoring peut se terminer soit avec sucées ou avec un échec. Le succès est réalisé si l'objectif final spécifié par la règle événementielle *ER* est atteint. Ce cas de figure est obtenu si on arrive à exécuter tous les services du dernier niveau du sous graphe associé à l'événement détecté. Par contre, on considère que le monitoring est échoué si le seuil limite fixé à la recomposition est dépassé alors que l'objectif spécifié dans la règle *ER* n'est pas atteint. A la fin, la valeur du paramètre *EventMonitoringState (EMS)* indique l'état final du processus de monitoring. En cas d'un monitoring réussi (i.e. *EventMonitoringState = True*), le sous graphe *SG* correspondant sera associé à l'événement détecté et sera enregistrée dans la base de données des sous graphes (*SGDB*).

Services Monitoring Algorithm (SeMA)

Inputs: OccurredEvent: OE, Replanning Threshold: RT
Outputs: Event Monitoring State: EMS

Begin

```
1.  Goal=OE.getEventRule.getGoal
2.  SubGraph=getEventPlan(OE.getEvent)
3.  If (SubGraph.size=0) then
4.     SubGraph =SubGraphExtraction(Goal)
5.  End if
6.  EventMonitoringState=True, graphLevelIndex=0, NumberOfReplanning=0
7.  While ((EventMonitoringState=True)∧(graphLevelIndex < SubGraph.size())) do
8.    ServicesToMonitore = SubGraph.getServicesOfLevel(graphLevelIndex)
9.    GraphLevelMonitoringState=True
10.   While ((GraphLevelMonitoringState=True)∧(ServicesToMonitore≠∅)) do
11.     AS= ServicesToMonitore.getService
12.     If (AS.monitored=False) then
13.        AS.monitored=True
14.        RequiredMessages = AS.getRequiredMessages()
15.        AS.Response=CSS (AS, RequiredMessages, OE.getLocation)
16.        ServicesToMonitore.delete(AS)
17.        If (AS.Response =False) then
18.          Remove AS from SubGraph and GlobalGraph
19.          Mark AS as unavailable in ASDB
20.          Start process for GlobalGraph update from graphLevelIndex
21.          If (∃ ASi ∈ ASDB such as (ASi^in ⊆ TSM)∧(ASi^out ⊇ RequiredMessages)) then
22.            Replace AS by ASi in SubGraph
23.            ServicesToMonitore.add(ASi)
24.          Else
25.             GraphLevelMonitoringState=False
26.             If (NumberOfReplanning<RT) then
27.                SubGraph =SubGraphExtraction(Goal)
28.                Increment NumberOfReplanning
29.             Else
30.                EventMonitoringState=False « fail in handling event (OE.getEvent) »
31.             End if
32.          End if
33.        End if
34.     End if
35.   End while
36.   If (GraphLevelMonitoringState=False) then
37.      Increment graphLevelIndex
38.   Else
39.      graphLevelIndex=0
40.   End if
41. End while
42. If (EventMonitoringState=True) then
43.   Save SubGraph for the Occurred Event OE in EDB
44.   Send the Goal to the event notification and devices control module
45. Else
46.   Send the Occurred Event OE to the event notification and devices control module
47. End if
```

End

Figure 6.24 : Algorithme de monitoring de services

L'objectif obtenu à la fin du monitoring sera également envoyé au module de notification et de contrôle des dispositifs du Framework *FrEvASeC*. Ce module vérifie les valeurs contenues dans les messages contenus dans l'objectif obtenu à celles spécifiées par la règle événementielle *ER*. Dans le cas ou il ya une correspondance entre ces valeurs, l'utilisateur concerné sera notifié par l'événement détecté et l'objectif obtenue. Dans le cas contraire, ou bien dans le cas d'un monitoring écoué (i.e. *EventMonitoringState = False*), l'utilisateur peut être notifié juste par le nom de l'événement détecté.

6.5.4. Niveau inférieur de monitoring de service

Le niveau inférieur de monitoring de services gère les défaillances qui surviennent lors de la phase d'exécution des services concrets. Son objectif principal étant de trouver pour chaque service abstrait appartenant au sous-graphe *SG* un meilleur service concret en termes de qualité de service (*QoS*) fournie. Pour ce faire, ce niveau de monitoring se base sur l'algorithme de sélection de services concrets (*CSS*), décrit dans la **section 6.4.4**, qui intercale la sélection et l'apprentissage. L'algorithme *CSS* lui-même est basé sur l'algorithme d'invocation de services concrets (*CSI*) décrits dans la **section 6.4.3**. L'algorithme *CSI* invoque les services concrets et met à jour concrètement leur qualité dynamique (*DQoS*). Cet algorithme essaie d'apprendre davantage sur les probabilités de réponse (*PR*) des services concrets en utilisant l'apprentissage Bayésien. Lorsque la réponse obtenue après l'invocation d'un service concret est *positive*, le niveau inférieur de monitoring retourne immédiatement cette réponse au niveau supérieur de monitoring de services décrit précédemment. Dans le cas contraire, l'algorithme de sélection *CSS* tente de remplacer immédiatement le service concret échoué par le meilleur service concret du service abstrait courant, i.e. en cours de sélection. Une fois que le seuil d'apprentissage est dépassé sans trouver aucun service concret adéquat, le niveau inférieur de monitoring remonte l'échec au niveau supérieur de monitoring. Ce dernier cherche alors à remplacer le service abstrait échoué par la procédure décrite dans la section précédente. Dans le niveau inférieur de monitoring, l'algorithme d'invocation de services concrets *CSI* joue le rôle d'un invocateur de service (*service invoker*) qui maintient et met à jour la table *TSM*. En Effet, les valeurs des messages reçus des services concrets déjà invoqués sont enregistrées dans la table *TSM*. Ces valeurs peuvent être utilisées afin d'invoquer d'autres services concrets contenues dans les services abstraits du sous graphe *SG*. Ainsi, à ce niveau de monitoring de services, il est possible de créer un graphe exécutable de services concrets. Ce graphe est obtenu à partir de la substitution des services abstraits du sous graphe *SG* par leurs services concrets sélectionnés correspondants.

6.6. Conclusion

Dans ce chapitre, nous avons détaillé le fonctionnement de cinq modèles principaux orientés services à savoir : le modèle de découverte de services, le modèle de classification de services, le modèle de composition de services, le modèle de sélection de services et enfin le modèle de monitoring de services. En réalité, d'après leur description techniques, ces cinq modèles correspondent aux six modèles orientés services cités dans la **section 2.6.2 du chapitre 2** à savoir : *le modèle d'évaluation de la QoS et du contexte, le modèle de matching des services, le modèle de sélection des services, le modèle de découverte des services, le*

modèle de composition des services, et *le modèle de monitoring des services*. En effet, notre modèle de sélection de services englobe les deux modèles suivants : *le modèle d'évaluation de la QoS et du contexte* et *le modèle de sélection des services.* Notre modèle de classification de services englobe, quant à lui, *le modèle de matching des services.* Les trois modèles: *architecture générale*, *modèle des connaissances et modèle de communication* ont été détaillés dans le chapitre précédent, tandis que le *modèle d'évaluation* sera bien détaillé dans le chapitre suivant. Ainsi, tout compte fait, on retrouvera nos dix modèles orientés services cités dans la **section 2.6.2 du chapitre 2. La table 6.2** ci-dessous donne un récapitulatif des méthodes et techniques utilisées par les modèles orientés services présentés dans ce chapitre.

Modèles	Descriptions
Modèle d'évaluation de la QoS et du contexte	- Evaluation de la qualité de service statique - Evaluation de la qualité de service dynamique - Evaluation de la qualité de service globale - Mise à jour des paramètres de qualité dynamique - Apprentissage Bayésien sur les probabilités de réponse des services
Modèle de matching de services	- Descriptions des concepts dans une ontologie - Similarité sémantique entre les concepts - Similarité sémantique entre les messages fournis et requis par les services concrets - Critère d'équivalence fonctionnellement entre les services concrets - Identification et classification des services fonctionnellement équivalents
Modèle de sélection de services	- Sélection d'un service concret appartenant à un service abstrait selon sa qualité globale. - Invocation des services concrets - La sélection de services est augmentée par un mécanisme d'apprentissage Bayésien.
Modèle de découverte de services	- Architecture à plusieurs répertoires locaux de services - Répertoire global des services concrets - Mécanisme de souscription /notification - Mode actif et mode passif
Modèle de composition de services	- Représentation graphique des services abstraits - Règles logiques de raisonnement sur les entrées et les sorties des services abstraits. - Représentation d'une composition de services sous formes d'un graphe abstrait - Planification à base de règles pour la construction d'un graphe de composition de services. - Construction automatique d'un graphe global de services on offline. - Extraction automatique des sous graphes de services en online
Modèle de monitoring de services	- Deux mécanismes de substitution de services : un mécanisme de substitution local et un mécanisme de substitution global. - Approche par niveaux de monitoring - Niveau initial de monitoring - Niveau supérieur de monitoring - Niveau inférieur de monitoring

Table 6.2 : Tableau récapitulatif de la stratégie de composition de services

Les six modèles présentés dans ce chapitre visent la réalisation, dans un cadre orienté service, des exigences des systèmes intelligents ambiants détaillées dans la **section 1.4 du chapitre 1** à savoir : *l'intelligence, la sensibilité au contexte, l'auto-adaptation, la couche d'abstraction, la transparence et le passage à l'échelle*. La qualité des réponses fournies par chaque modèle est estimée par les huit derniers points que nous avons identifiés dans la **section 4.3.2 du chapitre 4** à savoir : *Evaluation de la QoS et du contexte (QCE- QoS and Context Evaluation)*, *Gestion des incertitudes (UM- Uncertainty Management)*, *Découverte dynamique de services (DSD- Dynamic Service Discovery)*, *Classification dynamique de services (DSC- Dynamic Service Classification)*, *Sélection dynamique de services (DSS- Dynamic Service Selection)*, *Composition automatique et dynamique de services (ADSC- Automatic and Dynamic Service Composition)*, *Auto-adaptation (SA- Self Adapting)*, et *Gestion spontanés des événements (SEH- Spontanious Events Handling)*. Ainsi, la **table 6.3** ci-dessous donne une estimation de la qualité des modèles présentés dans ce chapitre selon les huit points précédents.

Critère de qualité des modèles	Prise en compte du critère
Evaluation de la QoS et du contexte	✓
Gestion des incertitudes	✓
Découverte dynamique de services	✓
Classification dynamique de services	✓
Sélection dynamique de services	✓
Composition automatique et dynamique de services	✓
Auto-adaptation	✓
Gestion spontanés des événements	✓

Table 6.3 : Tableau de la qualité estimée des modèles proposés

CHAPITRE 7

Mise en œuvre et évaluation des performances

7.1. Introduction

Ce chapitre est dédié à la mise en œuvre et à la validation expérimentale des modèles de composition, de sélection et de monitoring des services proposés dans les **chapitres 5 et 6**. Son objectif principal est de montrer l'intérêt et la faisabilité de notre Framework *FrEvASeC* dans un environnement ambiant réel. Pour ce faire et compte tenu des équipements et des services disponibles au sein du laboratoire LISSI, nous avons, d'abord, mis au point un environnement ambiant prototypique sous forme d'une maison intelligente dans laquelle sont déployés l'ensemble de ces dispositifs et services. Nous avons, par la suite, implémenté plusieurs types de scenarii afin de contrôler cet espace intelligent et d'apporter les services nécessaires aux personnes qui s'y trouvent. Ces scénarii touchent à plusieurs domaines d'assistance et de maintien de personnes à domicile. Il s'agit notamment de personnes âgées à forte dépendance qui nécessitent une surveillance permanente et particulière de l'évolution de leurs états de santé. Cette surveillance permet d'anticiper et de réduire les risques d'accidents liés au vieillissement et d'effectuer une prévention médicale suite à une évolution de l'état de fragilité de la personne âgée.

Ce chapitre est organisé en trois parties de la manière qui suit. Dans la première partie, nous décrivons la plateforme ubiquitaire développée au laboratoire LISSI dans le cadre de ces travaux de thèse. Dans cette partie, nous décrivant, tout d'abord, les différents dispositifs constituants la plateforme matérielle. Ensuite, nous détaillons les différentes parties de la plateforme logicielle, entre autre, l'ontologie des connaissances manipulées, les services abstraits et concrets ainsi que les messages échangés entre ces services. Dans la deuxième partie, nous présentons l'implémentation de plusieurs scenarii d'assistance d'une personne à domicile. Ces scénarii correspondent à des configurations différentes de services et ils sont générés automatiquement suite à la détection de certains événements déclenchant dans l'environnement ambiant. Chaque configuration de services est représentée par un sous graphe qui est extrait automatiquement du graphe global de tous les services disponibles. Nous présentons aussi, en détail, l'exécution et le monitoring de certains scénarii. Enfin, dans la dernière partie, nous montrons, à travers une série de test d'évaluation des performances, l'apport et l'intérêt des différents algorithmes qui constituent le noyau de notre Framework *FrEvASeC*.

7.2. Implémentation prototypique sur la plateforme ubiquitaire du LISSI

7.2.1. Plateforme matérielle

Cette plateforme a été mise en œuvre pour des besoins de validation des modèles de composition, de sélection et de monitoring des services développés dans les **chapitres 5 et 6**. Les scénarii exposés dans la suite de ce chapitre ont été implémentés en exploitant cette plateforme qui comprend différent composants :

- Un robot compagnon qui intègre des services divers : localisation via Karto, déplacement/navigation via Karto, reconnaissance vocale, vidéoconférence, agenda, mail, météo et monitoring de prise de médicaments;
- Des dispositifs d'affichage : téléphone portable de type Smartphone, moniteur de PC, écran d'affichage du robot compagnon ;

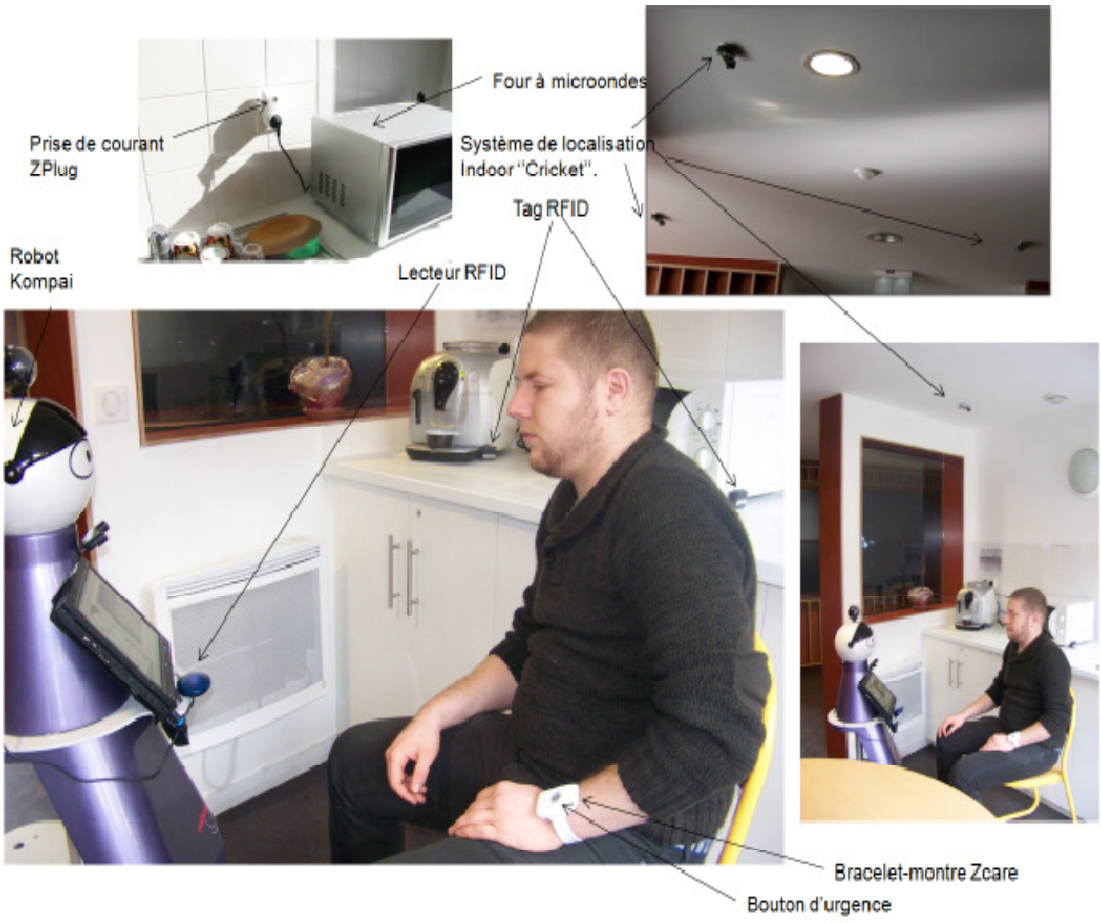


Figure 7.1 : Vue partielle de la plateforme ubiquitaire du Laboratoire LISSI.

- Un ensemble de capteurs permettant d'observer des évènements liés au contexte d'une personne âgée à domicile :
  - Un lecteur RFID longue porté (de 1 à 8 mètres) permettant d'identifier une entité portant un tag actif. Ce lecteur, embarqué sur le robot compagnon, permet aussi d'estimer la position grossière de l'entité en exploitant le système de localisation du robot ;

  - Un lecteur RFID courte portée (quelques centimètres), porté au poignet d'une personne, permet d'identifier une entité taguée et d'établir sa proximité par rapport à la personne ;
  - Des capteurs permettant de remonter l'accélération 3D et des informations d'ambiance (température, humidité, lumière, …) ;
  - Des capteurs de détection d'ouverture de portes, de détection de présence et de mesure d'intensité lumineuse;
  - Des détecteurs de monoxyde de carbone et de fumée ;
  - Un bracelet permettant de détecter la chute d'une personne, de mesurer son pouls et de remonter une alarme par appui sur un bouton d'urgence ;
  - Des caméras pour le tracking et la localisation ;
- Des actionneurs permettant de commander des équipements à distance :
  - Un actionneur de type TOR (Tout Ou Rien) permettant d'actionner une prise de courant à distance, et ainsi de mettre en marche un équipement ;
  - Des actionneurs dédiés permettant d'allumer/d'éteindre des ampoules électriques à distance;
- Un système de localisation indoor à base des capteurs Imote2 et des Crickets.

La **figure 7.1** donne une vue partielle de la plateforme ubiquitaire. Dans ce qui suit, nous donnons une description synthétique de chacun des composants de cette plateforme.

7.2.1.1. Le robot compagnon Kompai

Le robot Kompai développé par la société Robosoft, est équipé de plusieurs capteurs et actionneurs permettant d'assurer des fonctions essentielles telles que la planification de trajectoire, la navigation en environnement encombré, la cartographie, la localisation, l'interaction pour des tâches d'assistance au quotidien, **figure 7.2**. Il est équipé des composants suivants : deux caméras, une tablette-PC, un télémètre laser, des capteurs à ultrasons, des capteurs à infrarouge, des détecteurs de contact, deux moteurs, une batterie, une unité de contrôle embarquée et une interface Wifi.

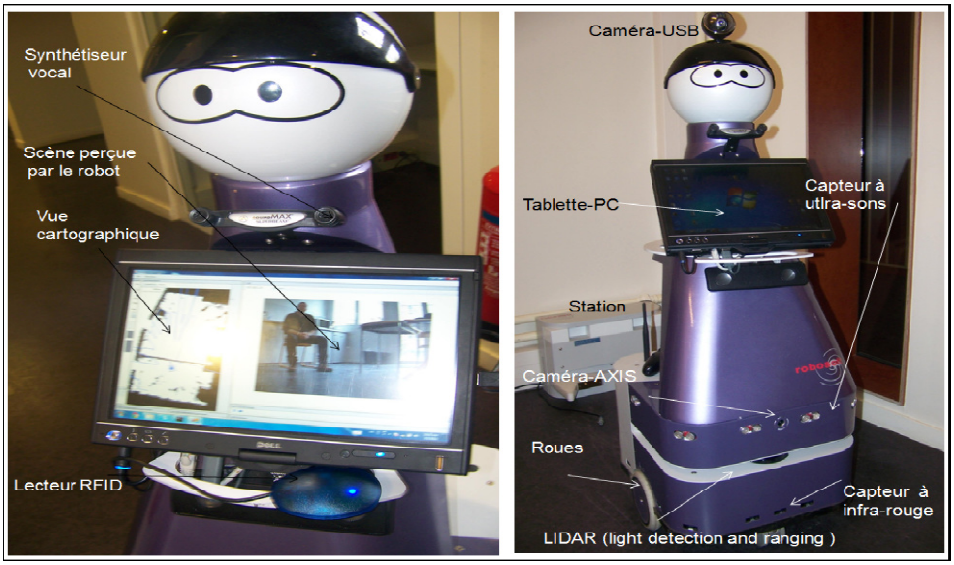


Figure 7.2 : Le robot Kompai.

Le robot Kompai embarque également une importante partie logicielle qui assure des fonctions de contrôle de bas niveau, jusqu'aux services de haut niveau à destination de l'utilisateur final. Ainsi, le robot propose des services tels que l'agenda, la messagerie Skype, la gestion des mails, la possibilité de faire ses courses en ligne, le contrôle par reconnaissance vocale, etc. L'architecture logicielle open source du robot, appelée RobuBOX est constituée de trois couches, **figure 7.3** :

- RobuBOX Services : Elle permet d'ajouter des services de plus haut niveau en utilisant l'intergiciel Microsoft Robotics Developer Studio (MRDS). Comme exemples de services, on peut citer : les services de cartographie, de navigation, de planification de trajectoire, etc. ;
- RobuBOX Lokarria : Elle consiste en un ensemble de services fournissant un accès à distance au robot ;
- RobuBOX PURE : Elle est chargée des contrôles de bas niveau (boucles de contrôle, machine à états, etc.) ;

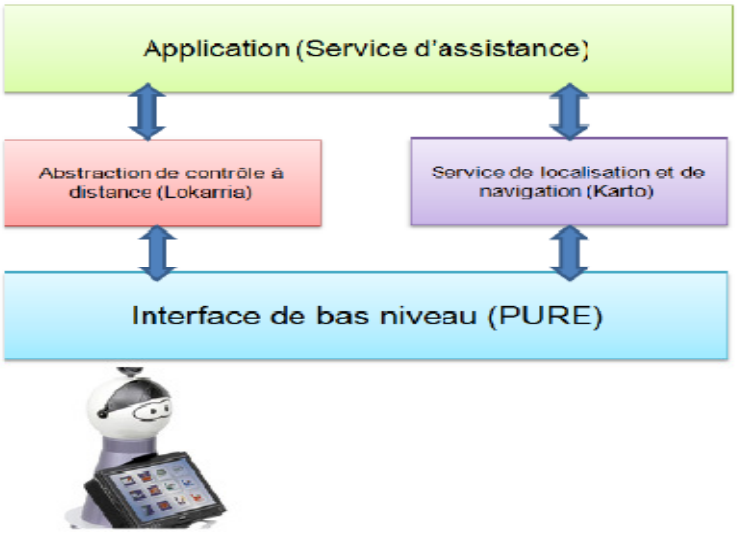


Figure 7.3 : Les trois couches de l'architecture robuBOX.

Si la quasi-totalité de la robuBOX est codée en C#, de nombreux services disposent d'interfaces REST qui permettent le contrôle du robot par des applications tierces, en utilisant le protocole HTTP. Le robot Kompai intègre le logiciel Karto qui est une suite logicielle incorporant des algorithmes avancés pour la cartographie et la navigation autonome des robots mobiles. Karto offre les fonctionnalités suivantes : La localisation, la cartographie par technique de SLAM, l'exploration, la planification de trajectoires et l'évitement d'obstacles. Le logiciel Karto est livré avec un module réseau qui permet la transmission de données distantes (odométrie, balayage laser) vers le module de cartographie (serveur SLAM).

7.2.1.2. Les capteurs Imote2

Pour les besoins de nos scénarii, nous avons utilisé des capteurs Imote2 de la plate-forme *Crossbow*. Un capteur Imote2 comporte trois principaux composants: *IPR2400/10*, *ITS400* et *IIB2400*. Tout d'abord, *IPR2400/10* est le principal composant qui gère les programmes embarqués grâce à un processeur XScale (320/416/520MHz PXA271) et une mémoire fournissant une taille de 32 Mo pour les programmes flashés et une taille de 32 Mo pour les

données. Le programme réside dans le nœud capteur même si ce dernier est éteint, et ce, contrairement aux données qui sont volatiles car elles sont traitées uniquement par les programmes en cours d'exécution. Le composant *IPR2400/10* supporte également les communications radio grâce à une antenne intégrée qui fonctionne suivant la norme Zigbee à une fréquence de 2,4 GHz et pouvant atteindre jusqu'à 30m en champ libre. Ensuite, *ITS400* est le composant qui se charge de la capture des données ambiantes suivantes: accélération, humidité, luminosité et température. L'accélération est mesurée suivant les trois axes x, y et z. Les valeurs de l'humidité sont comprises entre 0 et 100% RH avec une précision de +/-2 à 4%. Les valeurs de la luminosité sont codées sur 16 bits et peuvent être converties en lux. La température mesurée varie entre -40 et 120 °C avec une précision de +/-0,5 à 2 °C. Les valeurs de la température sont codées sur 14 bits. Enfin, le composant *IIB2400* permet la communication entre les nœuds Imote2 et le PC via le port USB. Il constitue un point d'accès JTAG par une communication série de 115 200 bauds (bits par seconde). Ce composant peut être fixé à n'importe quel composant *IPR2400/10* pour agir comme une station de base. Cette dernière se charge de collecter les données du réseau de capteurs et les remonter à l'ordinateur afin qu'elles soient utilisées par les différents services.

7.2.1.3. Les capteurs/actionneurs CLEODE

Toujours pour les besoins de nos scenarii, nous avons utilisé la gamme des capteurs/actionneurs de la société CLEODE. Il s'agit d'une plate-forme matérielle et logicielle basée sur le réseau sans fil Zigbee.

- **Prise de courant ZPlug :** La prise de courant ZPlug permet de commander tout type d'appareils 220V ne dépassant pas 3500W de consommation électrique. Elle fournit également une information sur la consommation électrique de l'appareil lorsque celui-ci est activé. Cette prise de courant embarque une application permettant la commutation d'un appareil électrique par l'intermédiaire d'une commande On/Off et la mesure de la consommation de ce même appareil.
- **Bracelet-montre ZCare :** Un bracelet-montre émet des alertes radiofréquences sur détection de défaillance de type :
  - Alerte manuelle par appui sur un bouton d'urgence ;
  - Détection d'un pouls hors norme : A partir de la mesure périodique du pouls et de la valeur moyenne fixée à l'initialisation, le bracelet peut émettre une alerte ;
  - Détection d'une chute : Le bracelet permet de détecter une chute à partir d'un profil d'activité fixé à l'initialisation.
- **Détecteur ZDoor :** Ce détecteur permet de détecter l'ouverture/fermeture d'une porte/fenêtre.
- **Système de commutation de lumière ZLight :** Il s'agit d'un commutateur de lumière qui permet de commander deux lampes d'une puissance maximale de 500W chacune.
- **Détecteur de présence ZMove :** Ce détecteur utilise un capteur infrarouge pour détecter des mouvements dans une pièce dans un rayon de 10 mètres maximum.
- **Détecteur ZGas** : Détecte toute présence de fumée ou de monoxyde de carbone (CO).
- **Détecteur ZLum** : Détecteur de luminosité.

7.2.1.4. Les Tags RFID

Cette section détaille les tags et le lecteur RFID (*Radio Frequency IDentification*) du constructeur *Wavetrend* **(Figure 7.4)**. Ils sont utilisés dans plusieurs applications potentielles, notamment la surveillance des mouvements des différents objets tagués (étiquetés par des tags RFID).

- **Lecteur RFID L-RX400:** Le rôle de ce lecteur consiste à détecter et lire tous les tags qui existent dans sa zone de couverture. Cette détection est basée sur le signal RSSI (*Received Signal Strength Indicator*) reçu des différents tags sur un rayon maximal de 7 m. L'interface de ce lecteur est reliée au PC via un câble USB standard (incorporé au lecteur). Grace à cette conformité au standard USB, ce lecteur peut être connecté à n'importe quel hub USB. Ce lecteur est complètement un dispositif *Plug and Play*. Il travaille en même temps que les tags actifs. Son design externe est conçu de telle manière qu'il augmente la valeur esthétique du bureau de l'utilisateur (ressemble bien à une souris). L'utilisateur peut spécifier un seul ou bien plusieurs tags à surveiller par le lecteur. Une DLL est fournie pour faciliter l'accès au lecteur. Ceci simplifie davantage l'interfaçage logiciel/matériel, et rend le lecteur complètement transparent.
- **Tag RFID TG1800:** Chaque tag RFID possède un identifiant unique attribué par le constructeur lors de sa fabrication. Le TG1800 est un tag actif qui émette continuellement un signal contenant son identifiant qui sera détecté par le lecteur RFID qui se trouve dans la porté de son signal. Cela, est à l'inverse des tags passifs qui n'émettent que sur la demande du lecteur. Les tags actifs TG1800 sont une nouvelle génération conçue pour être portés sur le poignet (la courroie du poignet est vendue séparément). Une batterie interne alimente le tag. Cette batterie qui n'est ni changeable ni rechargeable est conçue pour durer le plus longtemps possible (environ 2 ans). L'idée consiste à envoyer des signaux de très faible puissance mais à des taux de transmissions très élevés.

(a) (b)

Figure 7.4 : RFID *Wavetrend* :(a) lecteur RFID L-RX400, (b) tag RFID TG1800

7.2.1.5. Les caméras

Nous avons utilisé plusieurs caméras dans les besoins de nos scénarii. Dans cette section, nous décrivons la caméra PTZ 214. Cette dernière est une caméra réseau du constructeur *Axis*, compatible avec les protocoles Internet IPv4 et IPv6, et d'une haute résolution d'environ 720x576 pixels. Elle embarque un serveur accessible via le protocole http. Ce serveur répond à différentes requêtes http de contrôle et de configuration. Les requêtes sont typiquement des commandes adressées à la caméra soit pour la configurer sur certains paramètres spécifiques tels que la résolution d'images ou bien pour effectuer certaines actions telles que la rotation,

l'inclinaison, le zoom et la transmission d'une séquence d'images. Le flux envoyé est en format Motion-JPEG et MPEG-4 simultanés avec un zoom maximal de X18. Plusieurs niveaux d'accès utilisateur avec protection par mot de passe, filtrage d'adresses IP, cryptage https et authentification IEEE 802.1X. Elle est également compatible avec le concept de Qualité de Service (QoS) ce qui permet de réserver la capacité réseau allouer à la vidéo et de classer les opérations de surveillance essentielles par ordre de priorité sur un réseau QoS.

7.2.1.6. Système de localisation indoor

Localisation par les crickets : Cricket est un système de localisation indoor conçu à l'origine par le MIT, **figure 7.5**. Il s'agit d'un réseau de capteurs sans fil basse consommation qui fournit deux informations de localisation. La première représente l'identifiant de l'espace où se trouve l'entité à localiser : *Kitchen*, *Living_Room*, *Room*, etc. La seconde information représente la position courante de l'entité en coordonnées cartésiennes 3D.

Figure 7.5 : Le système de localisation indoor "Cricket".

La façon la plus courante d'utiliser le système de localisation Cricket, consiste à déployer des balises de transmission (*Cricket beacons*) sur les murs ou les plafonds, et d'attacher la balise d'écoute (*Cricket listener*) à une entité mobile (*Personne, objet mobile, etc.*) dont la localisation doit être déterminée, **figure 7.6**. La position de l'entité mobile est ensuite estimée à partir de la mesure de la différence entre le temps de propagation des ondes RF et des ondes ultrasonores. A chaque fois que la balise d'écoute intercepte une information des balises de transmission, elle infère les coordonnées de sa position en se basant sur les distances par rapport aux balises de transmission (dont les positions sont connues a priori).

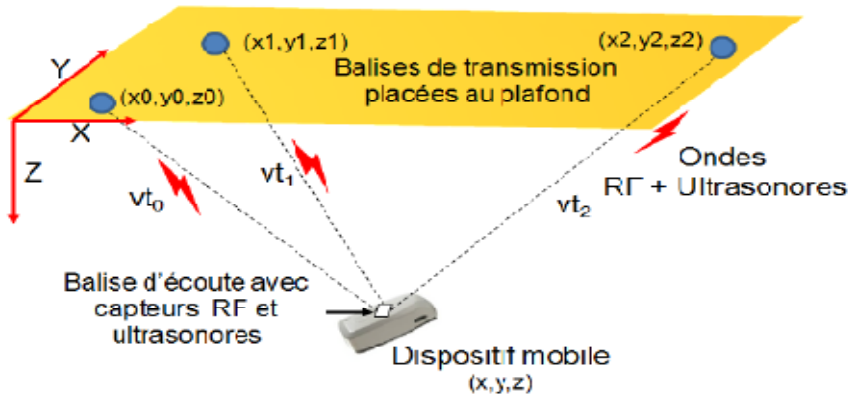


Figure 7.6 : Calcul de localisation par la méthode de triangulation.

Localisation par les capteurs Imote2 : Cette approche de localisation préconise l'utilisation du signal RSSI (*Received Signal Strength Indicator*) comme information potentielle pour se localiser. Cette localisation ne nécessite pas de matériel spécialisé autre que les interfaces communes du réseau sans fil ayant la capacité de mesurer la puissance du signal reçu. L'idée consiste d'abord à affecter à chaque objet de l'environnement ambiant un nœud capteur capable d'émettre et de recevoir des messages radios. La communication sera donc possible entre ces différents objets et l'environnement deviendra ainsi un vaste réseau de capteurs sans fil. Ensuite, chaque personne présente dans cet environnement sera aussi munie d'un nœud capteur afin de communiquer avec les autres capteurs. Le système de localisation se base sur la technique des empreintes radios. Cette technique exploite la relation entre un endroit précis et les valeurs mesurées de la puissance du signal radio RSSI à cet endroit **[303].** Dans cette approche, chaque emplacement géographique est défini par une signature radio qui représente l'ensemble des mesures empiriques du RSSI effectuées à cet emplacement. Ainsi, une base de données des signatures radios avec les emplacements correspondants est définie au préalable lors d'une phase d'apprentissage offline. Dans la phase online, un nœud mobile peut estimer sa position par l'acquisition d'une signature radio, puis la comparer aux signatures connues dans la base de données. La disposition du réseau de capteurs déployé au niveau du laboratoire LISSI est montrée sur les deux images de la **figure 7.7**.

Figure 7.7 : Disposition des capteurs de localisation au laboratoire LISSI.

- **Phase d'apprentissage de la carte radio :** Chaque nœud fixe (f) du réseau collecte d'abord un vecteur contenant les RSSI par rapport à tous ses voisins fixes. Ensuite, ce vecteur est envoyé à la station de base. Enfin, tous les vecteurs RSSI collectés sont sauvegardés dans une matrice d'apprentissage de dimension n1*n1 (n1: nombre de capteurs du réseau). Cette matrice initiale est augmentée par des nœuds capteurs fictifs correspondants aux différentes positions prises par le robot mobile. Les vecteurs RSSI collectés par ce robot sont alors ajoutés à la matrice initiale. Soit n2 le nombre de ces vecteurs, donc la matrice sera de dimension (n1+n2)*n1. En effet, les nœuds fictifs seront ajoutés en ligne (pas en colonne) car les nœuds réels du réseau ne collectent pas des valeurs RSSI à partir des nœuds fictifs.

- **Phase de localisation :** Dans la phase de localisation, le nœud mobile (m) envoie un vecteur des RSSI par rapport à n1 nœuds réels du réseau. Ce vecteur possède la forme suivante : ($RSSI_{m/1}$, $RSSI_{m/2}$, ... $RSSI_{m/n1}$). Le système de localisation calcule d'abord la distance RSSI pour chaque ligne i de la matrice d'apprentissage précédente en utilisant la formule de distance euclidienne suivante :

$$Dist_i = \frac{1}{n1}\left(\sum_{j=1}^{n1}(RSSI_{m/j} - RSSI_{i/j})^2\right)^{1/2}$$

Par la suite, les k identifiants des k lignes correspondantes aux plus petites distances sont sélectionnés comme des nœuds voisins les plus proches du nœud mobile m. La position de ce nœud mobile est alors considérée comme le barycentre de ces k plus proches voisins. La pondération de l'apport de chaque voisin sélectionné est relative à la puissance du signal reçu de ce dernier.

La **figure** 7.**8** montre un exemple de la variation des signaux RSSI reçus par un capteur mobile à partir de 12 capteurs fixes. Ainsi, chaque signal de la **figure 7.8** correspond à un capteur fixe du réseau. Le nœud mobile échantillonne périodiquement ces signaux RSSI et les transmet au système de localisation. Ce dernier détecte alors les plus proches voisins du nœud mobile selon la puissance du signal reçu et estime sa position par rapport à ces voisins. L'erreur moyenne obtenue sur les différentes positions calculée est de +/- 1,2m. Ce résultat permet de bien connaitre la zone géographique (chambre, salon, etc.) de la personne localisée.

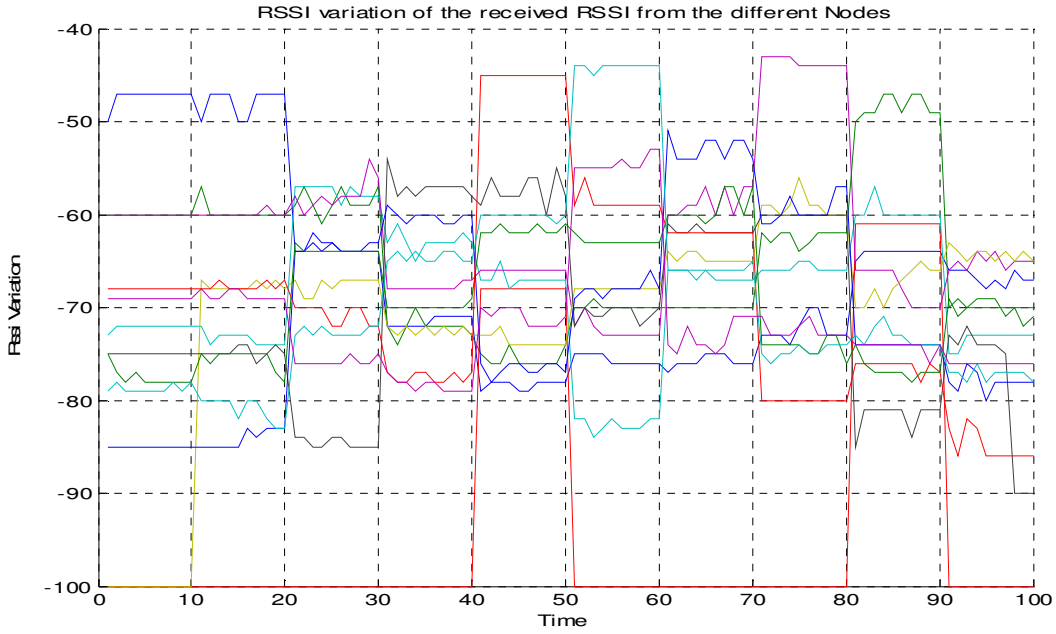


Figure 7.8 : Variation du signal RSSI reçu par le capteur mobile

7.2.2. Plateforme logicielle

Nous avons développé un ensemble de service afin d'exploiter les différents équipements illustrés dans la section précédente. Ces services partagent une ontologie commune qui décrit tous les paramètres de contexte contenus dans leurs messages d'entrée/sortie. De plus, chaque service développé appartient à l'une des quatre catégories de service suivante : les services sensibles aux événements, les services d'acquisition du contexte, les services de traitement du

contexte et les services actionneurs. Par ailleurs, afin de faciliter la gestion de tous les services disponibles, nous avons mis au point une application conviviale. Cette dernière permet de lancer l'exécution des services disponibles et de visualiser les résultats. De plus, cette application permet de lancer la composition, la sélection et le monitoring de services suite à la détection d'un événement. Les graphes de composition de services obtenus sont également visualisable par cette application.

7.2.2.1. Description de l'ontologie *AmbiOnt* des paramètres de contexte

L'ontologie de la **figure 7.9** décrits toutes les connaissances manipulées dans l'environnement ambiant expérimental, i.e. maison intelligente. Nous convenons d'appeler cette ontologie *AmbiOnt* (pour *Ambient Ontology*). Etant donné que l'ontologie *AmbiOnt* est de taille importante, sa description sur un seul diagramme sera complètement illisible et ne tient pas sur une page. Nous avons alors subdivisé cette ontologie en plusieurs sous ontologies de telle sorte qu'un concept racine d'une sous ontologie est marqué en couleur rouge tandis qu'un concept intermédiaire qui est développé dans une autre sous ontologie est marqué en couleur bleu. Dans toutes les sous ontologies décrivant l'ontologie *AmbiOnt*, les concepts terminaux sont marqués en couleur verte. Par ailleurs, le type d'une relation entre deux concepts de l'ontologie *AmbiOnt* est défini par un nom significatif. Nous convenons que lorsqu'aucun nom n'est indiqué sur une relation entre deux concepts, alors le type de cette relation est, par défaut, de type « *is a* ».

Tout d'abord, le concept *AmbientEnvironment* (**Figure 7.9**) représente le concept de base (i.e. la racine) de l'ontologie *AmbiOnt* et se réfère à tout l'espace intelligent étudié. Ce concept est relié à trois autres concepts essentiels à savoir : *Services*, *Entities* et *AmbientData*. En effet, la relation avec le concept *Services* indique qu'un environnement contient un ensemble de services. De même, la relation avec le concept *Entities* indique qu'un environnement contient un ensemble d'entités qui peuvent être soit des objets ou bien des personnes. Ensuite, la relation avec le concept *AmbientData* indique les conditions ambiantes qui règnent au sein de l'environnement. Par ailleurs, le concept *Services*, lui même, est relié au concept *Messages* par une relation d'utilisation. En effet, un service reflète les traitements (aspect dynamique) qui s'effectue au sein de l'environnement ambiant. Un service peut utiliser des messages en entrée et peut en produire d'autres en sortie. Ces messages peuvent être également échangés entre les différents services. Un message, à son tour, peut contenir des informations sur les entités (objets et personnes) et les conditions ambiantes (*AmbientData*) de l'environnement ambiant. Un message peut également contenir des informations particulières indiquant les différentes alertes générées (*AlertData*).

Ensuite, les trois concepts *AmbientData*, *AlertData* et *Entities* sont aussi décrits par des relations avec d'autres concepts à des profondeurs différentes de l'ontologie *AmbiOnt*. Premièrement, le concept *AmbientData* (**Figure 7.10**) renferme les paramètres d'ambiance échantillonnés (*SampledData)* et les paramètres d'ambiance analysés (*AnalyzedData*). Le concept *SampledData* se réfère aux valeurs brutes échantillonnées de plusieurs paramètres tels que la température, l'humidité, la luminosité, etc. Tandis que le concept *AnalyzedData* se réfère aux valeurs analysées de ces paramètres par rapport certaines normes spécifiées. Deuxièmement, le concept *AlertData* (**Figure 7.11**) renferme tout type d'alerte générées par

les services, notamment les alertes de santé, de conditions ambiantes, de sécurité, etc. Ces alertes sont utilisées afin de notifier les utilisateurs. Troisièmement, le concept *Entity* (**Figure 7.12**) est décrit par six éléments essentiels à savoir: l'identifiant (*ID*), la localisation (*Location*), le nom (*Name*), le type (*Type*), le nombre d'entités avoisinantes (*NumberOfNeighbors*) et les distances (*DistanceToNeighbors*) qui séparent l'entité avec ces propres voisins. Comme nous l'avons déjà mentionné, le concept *Entity* peut être soit une personne (*Person)* ou bien un objet (*Device).* En effet, les deux concepts *Person* et *Device* possèdent beaucoup de points en commun tels que l'identifiant, le nom, le type, la localisation et les voisins. Toutefois, le concept *Person* possède ses propres caractéristiques telles que : l'activité (*Activity*), la silhouette (*Silhouette*) et l'état de santé (*HealthStatus*). Cette dernière caractéristique (**Figure 7.13**) imbrique tous les paramètres de santé relatifs à une personne : le rythme cardiaque (*CardiacRate*), la température corporelle (*BodyTemperature*), la pression artérielle (*BloodPressure)*, etc.

Enfin, le concept *Device* (**Figure 7.14**) se réfère aux différents dispositifs de l'environnement ambiant notamment les dispositifs capteurs. Ces capteurs sont essentiellement au nombre de cinq: les capteurs simples (*SimpleSensors*), les capteurs visuels ou d'images (*ImageSensors*), les capteurs de son (*SoundSensors*), les capteurs de mouvements (*MotionSensors*) et les capteurs de tags (*RFIDTags*). Les capteurs simples remontent des données simples contenant des valeurs diverses : ambiance, santé, etc. Les capteurs visuels (i.e. caméras) remontent des données images. Les capteurs de son (i.e. microphones) remontent des données audio. Les capteurs de mouvements remontent des donnés contenant des valeurs de mouvements. Les capteurs de tags (lecteurs de tag RFID) remontent des valeurs des identifiants. Dans nos scenarii, les identifiants sont essentiellement transmis par les capteurs simples (*senorsID*) et les lecteurs de tag RFID (*RfidTagID*). Concernant les capteurs visuels, plusieurs types d'images (**Figure 7.15**) peuvent être transmises et traitées: image JPEG, image RGB, image contenant les blobs physiques des entités, et image contenant des formes géométriques (rectangles, cercles) qui encadrent des entités. Le reste des paramètres, qui représentent les feuilles (*concepts terminaux*) de l'ontologie *AmbiOnt*, sont décrits, en détail, dans la **table 7.1** de la **section 7.2.2.2**.

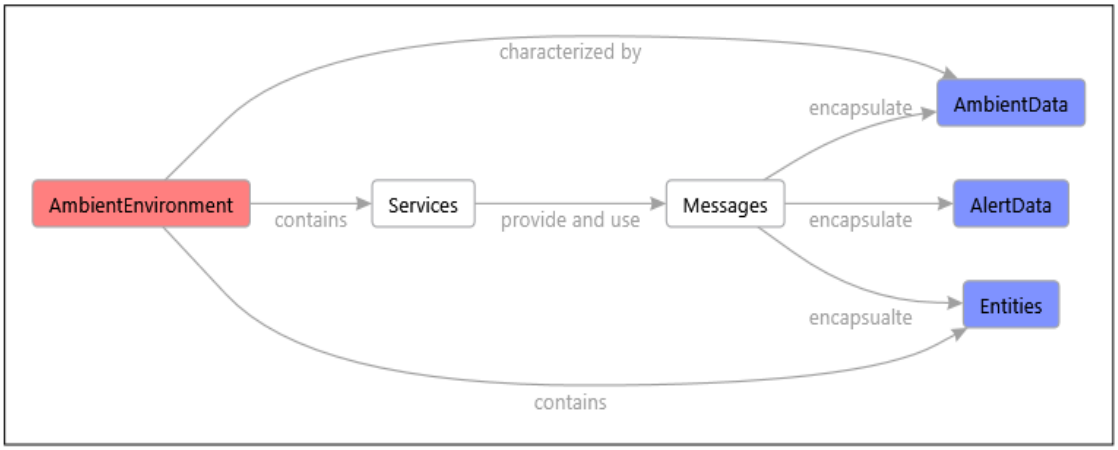


Figure 7.9 : Principaux concepts de l'ontologie « *AmbiOnt* »

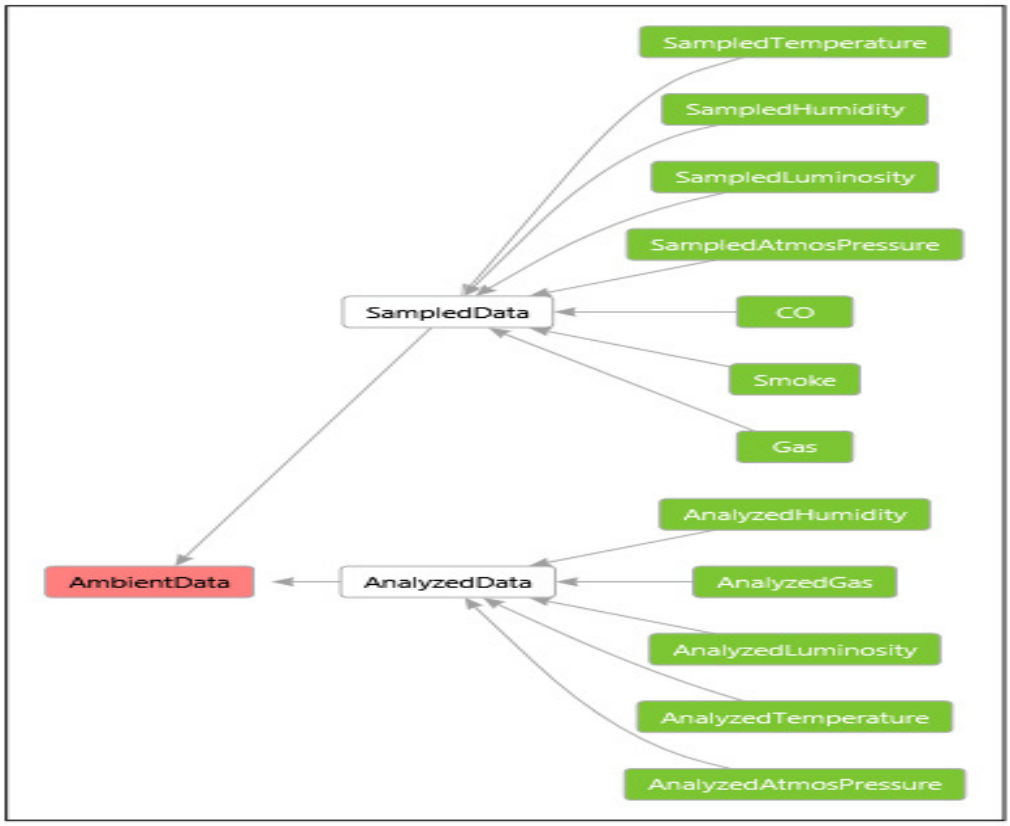


Figure 7.10 : Sous ontologie du concept « *AmbientData* »

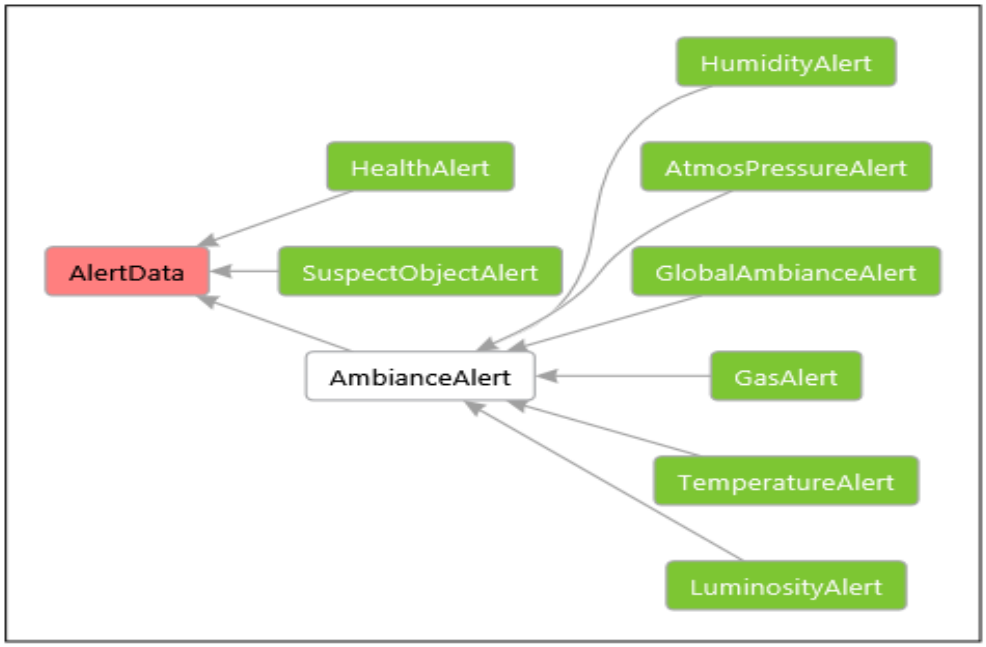


Figure 7.11 : Sous ontologie du concept « *AlertData* »

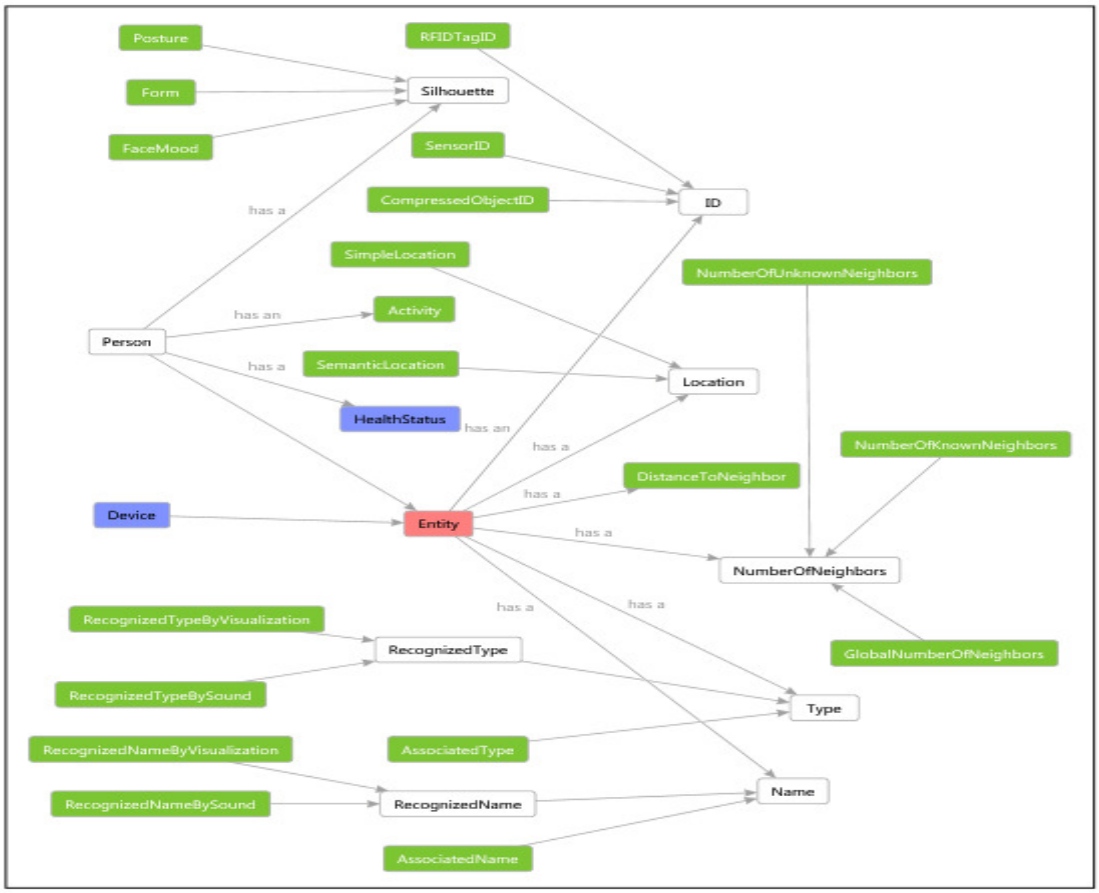


Figure 7.12 : Sous ontologie du concept « *Entity* »

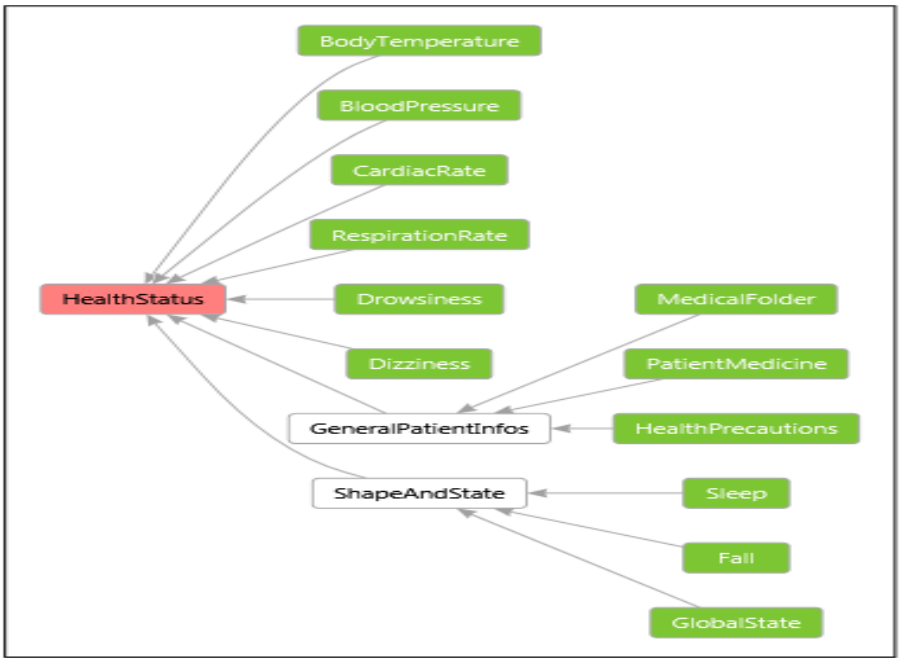


Figure 7.13 : Sous ontologie du concept « *HealthStatus* »

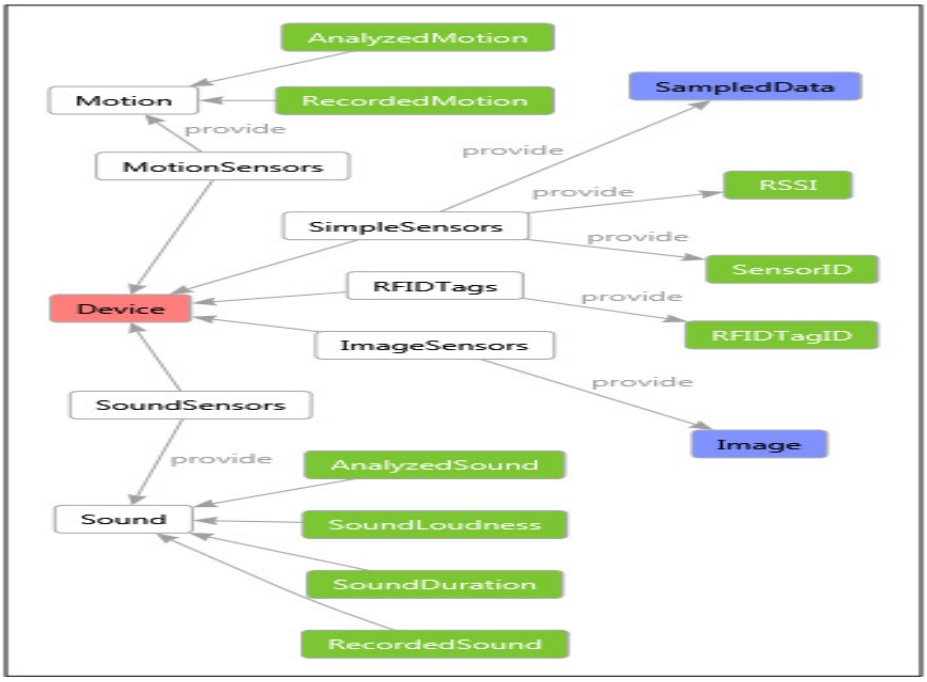


Figure 7.14 : Sous ontologie du concept « *Device* »

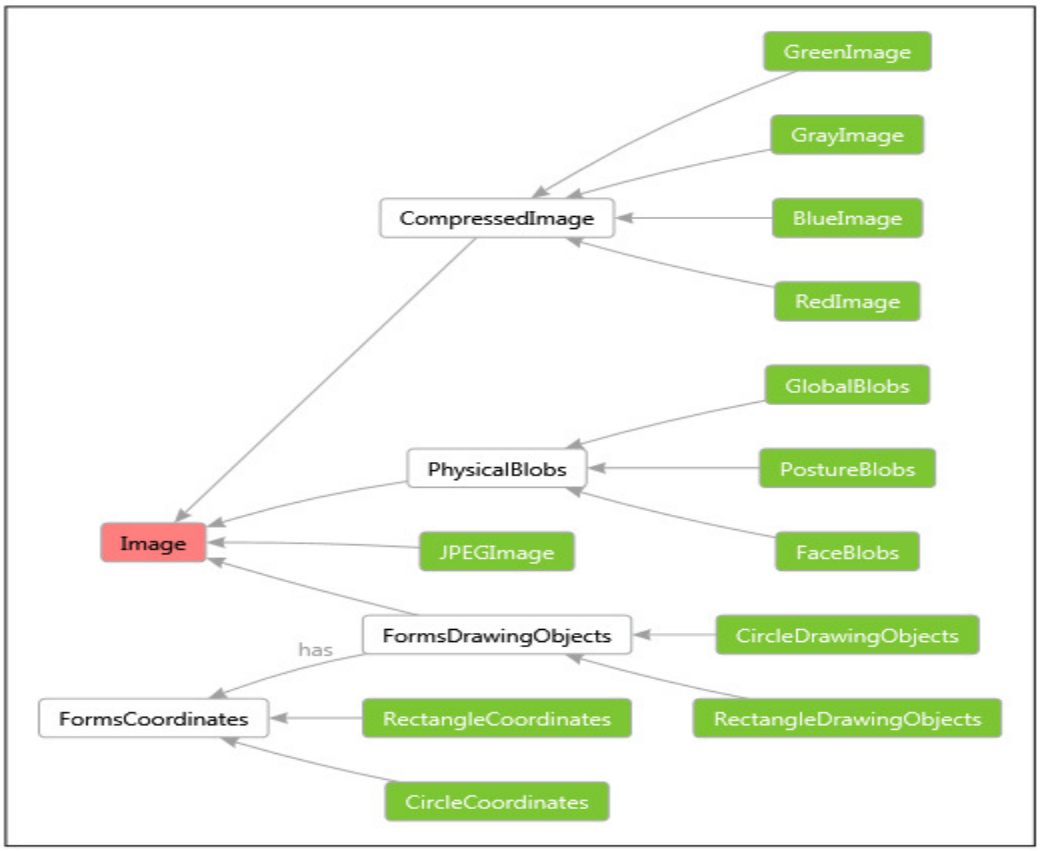


Figure 7.15 : Sous ontologie du concept « *Image* »

7.2.2.2. Description détaillée des concepts terminaux de l'ontologie *AmbiOnt*

La **table 7.1** ci-dessous décrit certains paramètres qui représentent les feuilles de l'ontologie. En général, ce sont les valeurs de ces paramètres qui sont encapsulées dans des messages puis échangées entre les services.

Parameter	Description	Valeurs
JPEGImage	Image en format JPEG	Ensemble de frame JPEG
GrayImage	Image en noir et blanc	Ensemble de frames en noir et blanc
RedImage	Image en rouge	Ensemble de frames en rouge
GreenImage	Image en vert	Ensemble de frames en vert
BlueImage	Image en bleu	Ensemble de frames en bleu
RectangleDrawingObjets	Un rectangle qui encadre un objet sur une image	Forme rectangulaire dessinée sur une image
RectangleCoordinates	Les coordonnés cartésiennes d'un rectangle	Quatre coordonnés cartésiennes x et y
CircleDrawingObjets	Un rectangle qui encadre un objet sur une image	Forme circulaire dessinée sur une image
CircleCoordinates	Les coordonnés cartésiennes du centre du cercle et son rayon.	Deux coordonnés cartésiennes x et y du centre et un rayon R
GlobalBlobs	Image contenant tous les blobs physiques détectés sur une image	Une image en noir et blanc tels que les pixels correspondants aux blobs sont en blanc et le reste de l'image étant en noir
PostureBlobs	Image ne contenant que les blobs physiques qui ressemblent à une posture	Une image en noir et blanc tels que les pixels correspondants aux blobs de la posture sont en blanc et le reste de l'image étant en noir
FaceBlobs	Image ne contenant que les blobs physiques qui ressemblent à une face	Une image en noir et blanc tels que les pixels correspondants aux blobs de la face sont en blanc et le reste de l'image étant en noir
Posture	Posture d'une personne	Debout, assis, allongé, accroupi, etc.
Form	Apparances physiques d'une personne	La taille, poids, cheveux, yeux, etc.
FaceMood	Humeur du visage d'une personne	Content, énervé, dormi, etc.
AssociatedName	Un nom associé à un identifiant soit d'un Tag RFID ou bien d'un capteur.	Nom du robot, nom de la personne, nom de l'objet compressible, et nom des autres objets
RecognizedNameBySound	Un nom reconnu par une reconnaissance vocale	Nom du robot, nom de la personne, nom de l'objet compressible, et nom des autres objets
RecognizedNameByVisualization	Un nom reconnu par une reconnaissance visuelle	Nom du robot, nom de la personne, nom de l'objet compressible, et nom des autres objets
AssociatedType	Un type associé à un identifiant soit d'un Tag RFID ou bien d'un capteur.	Robot, objet compressible, personne, et autres objets
RecognizedTypeBySound	Un type reconnu par une reconnaissance vocale	Robot, objet compressible, personne, et autres objets
RecognizedTypeByVisualization	Un type reconnu par une reconnaissance visuelle	Robot, objet compressible, personne, et autres objets
Activity	Activité d'une personne	Texte sémantique, par exemple "Ali est assis sur un fauteuil dans le salon", "Ali est dormi sur un lit dans la chambre", etc.
CompressedObjectID	L'identifiant d'un objet	F1 pour fauteuil1, F2 pour

	compressible	fauteuil2, Ch1 pour chaise1, etc.
RecordedMotion	Mouvements enregistrés aux passages des objets	Chaque enregistrement contient l'endroit de passage (ex. porte 1) et l'heure de passage
RecordedSound	Le son enregistré	Chaque enregistrement contient le fichier audio du son, l'endroit et l'heure d'enregistrement
SoundLoudness	amplitude du son	Ensemble de valeurs indiquant les amplitudes successives du son ainsi que l'endroit et l'heure de leurs enregistrement
SoundDuration	la durée du son	Ensemble de valeurs indiquant les durée des amplitudes successives du son
SimpleLocation	Position	Coordonnées x, y d'une position
SemanticLocation	Endroit	Chambre1, chambre2, salon, cuisine, couloirs, etc.
AnalyzedMotion	Corrélation entre les différents Mouvements enregistrés	Texte sémantique, par exemple "deux objets sont entrés par la porte 1 à 9 :00 ", "un objet est sorti par la porte 2 à 10 :00", etc.
AnalyzedSound	Interprétation des amplitudes et des durées du son	Texte sémantique, par exemple "un bruit intense et long est détecté dans le salon entre 12 :00 et 13 :00", "une voix basse est détectée dans le couloir à 10 :00", etc.
SuspectObjectAlert	Indique s'il ya au moins un objet suspect	Vrai ou faux
Drowsiness	Indique la somnolence détectée sur une personne	Degré de sommeil : 0, 1, 2, 3 Le niveau 3 étant le sommeil profond
RespirationRate	Indique le rythme respiratoire d'une personne	Volume d'air mobilisé à chaque cycle respiratoire, pendant une expiration ou une inspiration normale
BodyTemperature	Température corporelle de la personne	Température en C°
Fall	Indique si une personne a chuté ou non	Vrai ou faux
CardiacRate	Indique le rythme cardiaque de la personne	Electro cardiogramme
GlobalState	Indique l'état de santé global d'une personne	Excellent, très bon, bon, moyen, mauvais, critique
HealthAlert	Indique le niveau d'alerte sur l'état de santé d'une personne	0, 1, 2, 3 Le niveau 3 étant le plus critique
Sleep	Indique si une personne est dormie ou non	Pas de sommeil, début de sommeil, sommeil léger, sommeil profond.
RSSI	Indique l'intensité du signal reçu d'un nœud capteur	Valeur entre 0 et -100
SensorID	Identifiant d'un nœud capteur	1, 2, 3, etc.
RFIDTagID	Identifiant d'un tag RFID	1, 2, 3, etc.
DistanceToNeighbours	Mesure la distance à un objet voisin	Valeur en Mètre
NumberOfKnownNeighbours	Le nombre d'objets voisins détectés et reconnus	Valeur entière
NumberOfUnknownNeighbours	Le nombre d'objets voisins détectés mais non reconnus	Valeur entière
GlobalNumberOfNeighbours	Le nombre total d'objets voisins	Valeur entière
SampledLuminosity	Indique l'intensité de la	Valeur en lux

	luminosité échantillonnée	
SampledHumidity	Indique le taux d'humidité échantillonnée	Valeur en pourcentage
SampledTemperature	Indique la température échantillonnée	Valeur en C°
SampledAtmosPressure	Indique la valeur de la pression atmosphérique	Valeur en Bar
Gas	Indique s'il y a du gaz dans l'air	Vrai ou faux
CO	Indique s'il y a du CO dans l'air	Le taux du CO dans l'air
Smoke	Indique s'il y a une fumée dans l'air	Le taux de la fumée dans l'air
AnalyzedLuminosity	Indique le niveau de la luminosité	Très bas, bas, insuffisant, moyen, bon, intense, très intense
AnalyzedHumidity	Indique le niveau de la l'humidité	Bas, moyen, élevé, très élevé
AnalyzedTemperature	Indique le niveau de la température	Bas, moyen, bon, élevé, très élevé
AnalyzedAtmosPressure	Indique le niveau de la pression atmosphérique	Bas, moyen, élevé, très élevé
AnalyzedGas	Indique la qualité de l'aire	Mauvais, moyen, bon
HumidityAlert	Indique le niveau d'alerte sur l'humidité	0, 1, 2, 3 Le niveau 3 étant le plus critique
TemperatureAlert	Indique le niveau d'alerte sur la température	0, 1, 2, 3 Le niveau 3 étant le plus critique
LuminosityAlert	Indique le niveau d'alerte sur la luminosité	0, 1, 2, 3 Le niveau 3 étant le plus critique
GasAlert	Indique le niveau d'alerte sur la qualité de l'aire	0, 1, 2, 3 Le niveau 3 étant le plus critique
AtmosPressureAlert	Indique le niveau d'alerte sur la pression atmosphérique	0, 1, 2, 3 Le niveau 3 étant le plus critique
GlobalAmbianceAlert	Indique le niveau d'alerte global	0, 1, 2, 3 Le niveau 3 étant le plus critique
BloodPressure	Indique la pression sanguine	Valeur de la pression sanguine
MedicalFolder	Dossier médical d'une personne	Historique médical de la personne (maladies, régimes, etc.)
PatientMedicine	Prises de médicament d'une personne	Horaire des dernières prises de médicament de la personne
HealthPrecautions	Précautions à prendre pour une personne	Texte contenant une liste de précaution à prendre
Dizziness	Les étourdissements d'une personne	malaises, vertiges, perte de connaissance, etc.

Table 7.1 : Description détaillée des concepts terminaux de l'ontologie *AmbiOnt.*

7.2.2.3. Description des services abstraits

Les différents paramètres décrits dans la **table 7.1** précédente sont d'abord encapsulés dans des messages, puis utilisés et échangés entres les différents services abstraits disponibles dans l'environnement. Ces services ainsi que leurs messages d'entrée/sortie respectifs sont alors décrits dans la **table 7.2** ci-dessous.

id	Service	Inputs	Outputs	Description
1	GetImages		M_1 < JPEGImage>	Return une séquence d'image capturées dans le champ de vision de la caméra
2	GradientsAndEdges	M_1 < JPEGImage>	M_3 <GrayImage>	Retourne une image simple en noir et blanc avec l'ensemble des objets contenus dans cette image.
3	RGBDemultiplixer	M_1 < JPEGImage>	M_4 <RedImage>, M_5 <GreenImage> , M_6 <BlueImage>	Retourne trois types d'image correspondants aux couleurs RGB : rouge, verte et bleu
4	Labelizer	M_3 <GrayImage>	M_7<RectangleDrawingObjets>, M_8 <RectangleCoordinates>	Retourne un ensemble de rectangles qui délimitent les objets de l'image et les cordonnées cartésiennes de ces rectangles.
5	BlobAnalyzer	M_2 <DecompressedImage >	M_7<RectangleDrawingObjets>,, M_9 <GlobalBlobs>	Retourne tous les blobs physiques détectés sur l'image ainsi que les rectangles associés aux objets de l'image
6	BlobAnalyzer1	M_3<GrayImage>	M_7<RectangleDrawingObjets>,, M_9 <GlobalBlobs>	Retourne tous les blobs physiques détectés sur l'image ainsi que les rectangles associés aux objets de l'image
7	BlobAnalyzer2	M_4 <RedImage>	M_7<RectangleDrawingObjets>, M_9 <GlobalBlobs>	Retourne tous les blobs physiques détectés sur l'image ainsi que les rectangles associés aux objets de l'image
8	BlobAnalyzer3	M_5 <GreenImage>	M_7<RectangleDrawingObjets>, M_9 <GlobalBlobs>	Retourne tous les blobs physiques détectés sur l'image ainsi que les rectangles associés aux objets de l'image
9	BlobAnalyzer4	M_6 <BlueImage>	M_7<RectangleDrawingObjets>, M_9 <GlobalBlobs>	Retourne tous les blobs physiques détectés sur l'image ainsi que les rectangles associés aux objets de l'image
10	PostureExtraction	M_9 <GlobalBlobs>	M_{10} <PostureBlobs>	Retourne tous les blobs physiques qui ressemblent à une posture
11	PostureConstruction	M_8<RectangleCoordinates>	M_{10} <PostureBlobs>	retourne les blobs des postures qui correspondent aux coordonnées des rectangles délimitant les objets, et ce, en se basant sur une base de données des blobs des postures.
12	PostureAndFaceExtraction	M_9 <GlobalBlobs>	M_{10} <PostureBlobs>, M_{11} <FaceBlobs>	Retourne les blobs physiques associés à la posture et au visage.
13	PostureRecognition	M_{10} <PostureBlobs>	M_{12} <Posture >	Retourne la posture de la personne

14	PostureAndFaceRecognition	M_{10} <PostureBlobs>, M_{11}<FaceBlobs>	M_{12} <Posture >, M_{13} <FaceMood>	Retourne la posture et l'humeur du visage de la personne
15	VisualIdentification1	M_{10} <PostureBlobs>, M_{11}<FaceBlobs>	M_{18} <RecognizedNameByVisualization>, M_{15}<RecognizedTypeByVisualization>, M_{30}<NumberOfUnknownNeighbours>, M_{29} <NumberOfKnownNeighbours>	Retourne le nombre d'objets connus et le nombre d'objets inconnus ainsi que les types et les noms d'objets reconnus par le mode visuel
16	VisualIdentification2	M_{10} <PostureBlobs>	M_{15}<RecognizedTypeByVisualization>, M_{18} <RecognizedName ByVisualization>, M_{28} < GlobalNumberOfNeighbours >	Retourne le nombre total d'objets détectés ainsi que les types et les noms d'objets reconnus par le mode visuel
17	VisualActivityRecognition1	M_{12} <Posture >, M_{18}<RecognizedNameByVisualization>	M_{20} <Activity>	Retourne l'activité de la personne reconnue par le mode visuel
18	VisualActivityRecognition2	M_{12} <Posture >, M_{13} <FaceMood>, M_{18} <RecognizedNameByVisualization>	M_{20} <Activity>	Retourne l'activité de la personne reconnue par le mode visuel
19	GetCompressedObjects		M_{21} <CompressedObjectID>	Retourne l'identifiants des objets qui ont subi une pression
20	GetDetectedMotion		M_{22} <RecordedMotion >	Retourne tous les mouvements enregistrés aux passages des objets
21	CompressedObjectIdentification	M_{21} <CompressedObjectID>	M_{25} < CompressedObjectID, AssociatedName, SemanticLocation>	Retourne les noms des objets qui ont subi une pression et leurs localisations sémantiques
22	GetRecordedSound		M_{23} <RecordedSound>, M_{24} <SoundLoudness, soundDuration>.	Retourne le son enregistré ainsi que les amplitudes du son et leurs durées respectives
23	SleepDetector		M_{32} < Drowsiness >	Retourne les somnolences détectées sur la personne
24	GetRespirationRate		M_{33} <RespirationRate>	Retourne l'état respiratoire de la personne
25	GetBodyTemperature		M_{34} <BodyTemperature>	Retourne la température corporelle de la personne
26	FallDetector		M_{35} <Fall>	Détecte les chutes de la personne
27	GetCardiacRate		M_{36} <CardiacRate>	Retourne le rythme cardiaque de la personne
28	MotionAnalyzer	M_{22} <RecordedMotion >	M_{26} <AnalyzedMotion>	Analyse les différents mouvements détectés
29	SoundRecognition	M_{23} <RecordedSound>	M_{19} <RecognizedNameBySound>	Reconnaissance vocale des personnes
30	SoundAnalyzer	M_{24} <SoundLoudness, SoundDuration>	M_{27} <AnalyzedSound>	Analyse l'amplitude et la durée du son pour détecter le bruit
31	SleepRecognition	M_{32} < Drowsiness>	M_{39} <Sleep>	détecte le sommeil de la personne
32	GlobalStateAnalyzer	M_{33} <RespirationRate>, M_{34} <BodyTemperature> , M_{36} <CardiacRate>, M_{64} < BloodPressure>	M_{37} <GlobalState>	Retourne l'état de santé global de la personne

33	MotionAndSoundSensing	M_{19} <RecognizedNameBySound>, M_{26} <AnalyzedMotion>, M_{27} <AnalyzedSound>	M_{28} < GlobalNumberOfNeighbours > , M_{29} <NumberOfKnownNeighbours>	Estime le nombre global d'objets voisins et leurs états respectifs connus ou inconnus
34	MotionSensing	M_{26} <AnalyzedMotion>	M_{28} < GlobalNumberOfNeighbours >	Estime le nombre global d'objets voisins
35	MoundSensing	M_{19} <RecognizedNameBySound>, M_{27} <AnalyzedSound>	M_{28} < GlobalNumberOfNeighbours >, M_{29} <NumberOfKnownNeighbours>	Estime le nombre global d'objets voisins et leurs états respectifs connus ou inconnus
36	HealthReport	M_{37} <GlobalState>, M_{65} < MedicalFolder> , M_{66} < PatientMedicine>	M_{38} <HealthAlert>	Retourne une alerte sur l'état de santé de la personne
37	SensedActivityRecognition	M_{25} < PressedObjectID, AssociatedName, SemanticLocation>, M_{19} <RecognizedNameBySound>, M_{17} <AssociatedName>, M_{39} <Sleepping>	M_{20} <Activity>	Retourne l'activité de la personne détectée combinant les modes des capteurs simples et des capteurs de sons
38	SuspectDetection1	M_{29}<NumberOfKnownNeighbours>, M_{19} <RecognizedNameBySound>, M_{18} <RecognizedNameByVisualization>, M_{17} <AssociatedName>	M_{31} <SuspectObjectAlert>	Retourne une alerte sur les objets suspects détectés
39	SuspectDetection2	M_{29} <NumberOfKnownNeighbours>, M_{19} <RecognizedNameBySound>, M_{17} <AssociatedName>	M_{31} <SuspectObjectAlert>	Retourne une alerte sur les objets suspects détectés
40	SuspectDetection3	M_{29}<NumberOfKnownNeighbours>, M_{19} <RecognizedNameBySound>	M_{31} <SuspectObjectAlert>	Retourne une alerte sur les objets suspects détectés
41	GetSensorID		M_{40} < SensorID, (SensorID, RSSI)>, M_{41} <SensorID>	Retourne l'identifiant des capteurs mobiles et le signal RSSI par rapport à leurs capteurs voisins
42	GetRfidTagID		M_{42} <RFIDTagID>	Retourne l'identifiant des tags RFID détectés
43	LocalizationByRSSI	M_{40} < SensorID, (SensorID, RSSI)>	M_{43} <SimpleLocation>	Retourne la localisation d'un capteur mobile et les objets qui l'avoisinent
44	TranslateRSSIToDistance	M_{40} < SensorID, (SensorID, RSSI)>	M_{44}<SensorID,(SensorID, DistanceToNeighbour)>	Retourne les distances qui séparent le capteur mobile par rapport à ses voisins
45	SensorIdentification	M_{41} <SensorID>	M_{14}<AssociatedType >, M_{17} <AssociatedName>	Retourne le type des objets ainsi que leurs noms détectés par le mode de capture simple
46	RfidTagIdentification	M_{42} <RFIDTagID>	M_{14}<AssociatedType >, M_{17} <AssociatedName>	Retourne le type des objets ainsi que leurs noms détectés par le mode RFID
47	LocalizationByTag	M_{42} <RFIDTagID>	M_{43} <SimpleLocation>	Retourne la localisation d'un Tag RFID et les objets qui l'avoisinent

48	SensorAndTagIdentification	M_{41} <SensorID>, M_{42} <RFIDTagID>	M_{14}<AssociatedType >, M_{17} <AssociatedName>	Retourne le type des objets ainsi que leurs noms détectés en combinant le mode RFID et le mode de capture simple
49	LocalizationByDistance	M_{44}<SensorID,(SensorID, DistanceToNeighbour)>	M_{43} <SimpleLocation>	Retourne la localisation d'un capteur mobile et les objets qui l'avoisinent
50	SemanticLocalization	M_{43} <SimpleLocation>	M_{45} <SemanticLocation>	Conversion de la localisation à une localisation sémantique
51	GlobalIdentification	M_{17}<AssociatedName>, M_{18}<RecognizedNameByVisualization>, M_{19} <RecognizedNameBySound>	M_{28} < GlobalNumberOfNeighbours >	Retourne le nom des objets détectés en combinant le mode visuel avec d'autres modes.
52	GetLuminosity		M_{46} <SampledLuminosity>	Retourne la luminosité échantillonnée
53	GetHumidity		M_{47} <SampledHumidity>	Retourne l'humidité échantillonnée
54	GetTemperature		M_{48} <SampledTemperature>	Retourne la température échantillonnée
55	GetAtmosPressure		M_{49} <SampledAtmosPressure>	Retourne la pression atmosphérique échantillonnée
56	GasDetector		M_{50} <Gas>	Détecte le gaz
57	CODetector		M_{51} <CO>	Détecte le taux de CO
58	SmokeDetector		M_{52} <Smoke>	Détecte le taux de la fumée
59	LuminosityAnalyzer	M_{46} <SampledLuminosity>	M_{53} <AnalyzedLuminosity>	Analyse le niveau de luminosité
60	HumidityAnalyzer	M_{47} <SampledHumidity>	M_{54} <AnalyzedHumidity>	Analyse le niveau de l'humidité
61	TemperatureAnalyzer	M_{48} <SampledTemperature>	M_{55} <AnalyzedTemperature>	Analyse le niveau de température
62	AtmosPressureAnalyzer	M_{49} <SampledAtmosPressure>	M_{56} <AnalyzedAtmosPressure>	Analyse le niveau de la pression atmosphérique
63	GasAnalyzer	M_{50} <Gas>, M_{51} <CO>, M_{52} <Smoke>	M_{57} <AnalyzedGas>	Analyse l'air à l'intérieur
64	HumidityNotifier	M_{54} <AnalyzedHumidity>	M_{58} <HumidityAlert>	Envoie une notification sur le niveau de l'humidité
65	TemperatureNotifier	M_{55} <AnalyzedTemperature>	M_{59} <TemperatureAlert>	Envoie une notification sur le niveau de la température
66	PartialAmbianceNotifier	M_{53}<AnalyzedLuminosity>, M_{54} <AnalyzedHumidity>, M_{55} <AnalyzedTemperature>	M_{58}<HumidityAlert>, M_{59}<TemperatureAlert>, M_{60} <LuminosityAlert>	Envoie une notification sur l'ambiance partielle : température, humidité et luminosité
67	GlobalAmbianceNotifier	M_{53} <AnalyzedLuminosity>, M_{54} <AnalyzedHumidity>,	M_{58} <HumidityAlert>, M_{59}<TemperatureAlert>,	Envoie une notification sur l'ambiance globale : température, humidité, luminosité,

		M_{55}<AnalyzedTemperature>, M_{56}<AnalyzedAtmosPressure>, M_{57} <AnalyzedGas>	M_{60}<LuminosityAlert>, M_{61} <GasAlert>, M_{62} <AtmosPressureAlert>	pression atmosphérique et l'air à l'intérieur
68	GasNotifier	M_{57} <AnalyzedGas>	M_{61} <GasAlert>	Envoie une notification sur l'état de l'air à l'intérieur
69	GlobalConditionsNotifier	M_{58}<HumidityAlert>, M_{59}<TemperatureAlert>, M_{60}<LuminosityAlert>, M_{61}<GasAlert>, M_{62} <AtmosPressureAlert>	M_{63}<GlobalAlert>	Envoie une notification sur les conditions globale à l'intérieur
70	GetBloodPressure		M_{64} < BloodPressure>	Retourne la pression sanguine de la personne
71	CheckMedicalFolder		M_{65} < MedicalFolder>	Retourne le dossier médical de la personne
72	CheckMedicine		M_{66} < PatientMedicine>	Retourne les dernières prises de médicament de la personne
73	HealthPrecautions	M_{33} <RespirationRate>, M_{34} <BodyTemperature>, M_{64} < BloodPressure>, M_{36} <CardiacRate>, M_{65} < MedicalFolder>, M_{66} < PatientMedicine>	M_{67} < HealthPrecautions>	Retourne un ensemble de précautions à suivre
74	DizzinessDetector	M_{64} < BloodPressure>, M_{35} <Fall>, M_{12} <Posture >	M_{68} < Dizziness>	Retourne l'état des étourdissements

Table 7.2 : Description détaillée des services abstraits disponibles.

7.2.2.4. Description de quelques services concrets

L'exécution d'un scénario nécessite l'invocation d'un ou plusieurs services concrets qui offrent des fonctionnalités différentes. Chacun de ces services possède son propre domaine d'acquisition par l'utilisation optimale de ses propres dispositifs. Par exemple, plusieurs services concrets pour mesurer la température ambiante sont associés pour chaque groupe de capteurs sans fil. Dans notre plateforme expérimentale, la salle1 est équipé de 12 capteurs sans fil. Chaque groupe de 4 capteurs est associé à un service concret différent. On obtient alors trois services concrets qui se chargent de mesurer la température pour la salle1. Ainsi, chacun de ces services fournit les différentes valeurs de la température reçues à partir de ses propres capteurs sans fil. Ces valeurs peuvent être agrégées par un autre service concret qui calcule et retourne une valeur moyenne de la température analysée. Voici un autre exemple de deux services concrets fonctionnellement équivalents: soient deux services de capture d'images. Le premier est associé à la caméra installée dans la salle1 tandis que le deuxième est associé à la caméra installée dans la salle2. Les deux services transmettent les images capturées dans leur champ de vision respectif. Ces deux services peuvent être sélectionnés en fonction du champ de vision requis. Par exemple, si le champ de vision concerne la salle1, le premier service s'exécute avec succès et l'autre échoue. Ainsi, les différents services abstraits illustrés sur le tableau précédent contiennent un ensemble de services concrets appartenant à des endroits différent de l'environnement ambiant. Chaque service concret est définit par sa *localisation* et ses paramètres de qualité statique et dynamique comme décrit dans le **chapitre 5**. Les six paramètres de qualité statique sont les suivants : *Cout monétaire (CM), Cout de traitement (CT), Cout de Communication (CC), Niveau de sécurité (NS), Fiabilité (FB) et Performances (PF)*. Tandis que les trois paramètres de qualité dynamique sont les suivants : *Disponibilité (AV), Temps de réponse (RT) et Probabilité de réponse (PR).* En raison de l'espace important nécessaire pour décrire tous les services concerts disponibles, nous nous limitons uniquement dans ce chapitre à la description des services concrets appartenant au service abstrait *GetImages*. Les autres services concrets se présentent de la même manière. Le service abstrait *GetImages* contient dix services concrets. Chaque service concret correspond à l'une des dix caméras déployées au niveau du laboratoire selon la configuration suivante : deux caméras au niveau du couloir, deux caméras au niveau de la première salle (*Salle1*), deux caméras au niveau de la deuxième salle (*Salle2*), deux caméras au niveau de la troisième salle (*Salle3*) et enfin deux caméras sont portées respectivement par les deux robots mobiles *Kompai* et *Pekee II.* La localisation de ces deux dernières caméras change en fonction de la mobilité des robots. On suppose qu'initialement, le robot *Kompai* se trouve dans le couloir et le robot *Pekee II* se trouve dans la salle 1. L'évaluation du paramètre de qualité *Performances (PF)* se base sur *la Résolution et le Frame rate* de chaque camera. La **table 7.3** montre les valeurs des différents paramètres de qualité de service de chaque service concret appartenant au service abstrait *GetImages*.

Service concret	Localisation	Qualité statique						Qualité dynamique		
		CM	CT	CC	NS	FB	PF	AV	RT	PR
SC1	Couloir	0.25	0.36	0.29	0.52	0.25	0.36	1	10.23	0.29
SC2	Couloir	0.34	0.90	0.66	0.21	0.77	0.78	1	2.34	0.37

SC3	Salle1	0.12	0.25	0.93	0.49	0.58	0.25	1	4.50	0.47
SC4	Salle1	0.66	0.34	0.61	0.21	0.63	0.53	1	2.30	0.72
SC5	Salle2	0.15	0.62	0.19	0.25	0.43	0.16	1	1.95	0.44
SC6	Salle2	0.32	0.11	0.67	0.91	0.34	0.37	1	5.32	0.34
SC7	Salle3	0.21	0.45	0.58	0.78	0.50	0.22	1	5.5	0.50
SC8	Salle3	0.80	0.30	0.69	0.59	0.75	0.21	1	6.30	0.82
SC9	Couloir	0.12	0.35	0.82	0.20	0.49	0.29	1	2.90	0.39
SC10	Salle1	0.34	0.11	0.89	0.91	0.34	0.37	1	5.32	0.34

Table 7.3 : Description des services concrets appartenant au service abstrait *GetImages.*

Comme nous l'avons mentionné dans la **section 6.4.2.1 du chapitre 6,** un utilisateur peut spécifier des poids pour indiquer ses préférences par rapport à certains paramètres de qualité de service. Dans ce travail, l'utilisateur accorde plus d'importance pour le paramètre *Performances (PF)* ($W_{PF} = 6$) et exige un niveau élevé pour le paramètre *Fiabilité (FB)* ($W_{FB} = 6$) pour assurer une meilleure qualité des images reçues. En outre, l'utilisateur souhaite minimiser le paramètre *Cout de Communication (CC)* ($W_{CC} = 5$) afin d'assurer une bonne vitesse de connexion. L'utilisateur souhaite également un *Niveau de sécurité (NS)* moyen ($W_{NS} = 3$). Les deux paramètres restant à savoir : *Cout monétaire (CM) et Cout de traitement (CT)* ont un poids égal à 1. En se basant sur ces poids et les valeurs des paramètres de qualité de services spécifiées dans la **table 7.3**, nous pouvons, à présent, estimer la qualité de service statique (*SQoS*), la qualité de service dynamique (*DQoS*) et la qualité de service globale (*QoS*) en utilisant, respectivement, les formules **(8), (11) et (12.1)** du **chapitre 6**. Le résultat de l'application de ces formules est mentionné sur **la table 7.4** qui ordonne les différents services concrets en fonction de leur qualité globale (QoS).

Classement	Service concret	SQoS	DQoS	QoS
10	SC1	0,33090909	0,02582369	0,00854529
02	SC2	0,65772727	0,11077844	0,072862
05	SC3	0,52136364	0,08545455	0,04455289
01	**SC4**	**0,52909091**	**0,21818182**	**0,11543802**
06	SC5	0,27318182	0,14915254	0,04074576
09	SC6	0,48954545	0,05379747	0,02633631
07	SC7	0,46454545	0,07692308	0,03573427
03	SC8	0,54909091	0,11232877	0,0616787
04	SC9	0,44772727	0,1	0,04477273
08	SC10	0,54045455	0,05379747	0,02907509

Table 7.4 : La *QoS* des services concrets appartenant au service abstrait *GetImages.*

7.2.2.5. Interface graphique pour visualisation et test des services disponibles

Afin de faciliter la gestion de tous les services et les équipements disponibles, nous avons mis au point une application qui fourni plusieurs fonctionnalités à savoir : la création d'un service pour un dispositif, la visualisation de la description d'un service existant et le test de bon fonctionnent (exécution) des services existants. Cette application permet également de lancer la composition, la sélection et le monitoring des services suite à une détection d'un éventuel événement. Les différents graphes de composition de services ainsi que le résultat de leurs exécutions sont aussi visualisables à travers cette application. La **figure 7.16** montre un exemple d'un service concret qui retourne une image après son exécution.

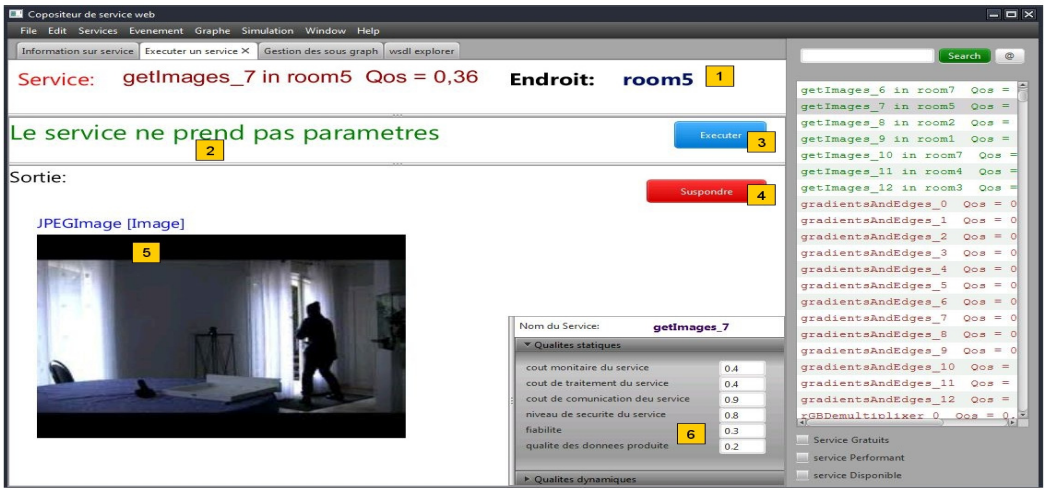


Figure 7.16 : Interface graphique pour test et visualisation des services

7.3. Illustration du Framework *FrEvASeC* sur des scenarii réels

7.3.1. Construction du graph global AGoSC

Le graphe global *AGoSC* de tous les services abstraits disponibles est montré sur **la figure 7.17**. Ce graphe est établi en offline par l'algorithme de construction du graphe global (*AGoSC Construction Algorithm*) afin de réduire le temps de réponse à un événement. Selon l'objectif spécifié, un sous graphe est extrait par la recherche de tous les prédécesseurs du service qui fourni le ou les messages spécifiés dans cet objectif. Les avantages de notre solution dans le contexte d'une maison intelligente sont multiples: 1) Focaliser sur l'événement en activant juste un sous graphe du graphe global des services, 2) Diminuer le nombre de fausse alerte envoyée à l'utilisateur (moins intrusif) grâce à l'analyse effectuée à plusieurs niveau du graphe de composition, 3) Reprise après panes et assurer la continuité du service en explorant les différentes chemins alternatifs, et 4) Assurer la réalisation du but global spécifié et informer l'utilisateur dans le cas d'impossibilité d'atteindre ce but. Ainsi, le déroulement de notre algorithme de construction sur les services abstraits disponibles pour l'acquisition est l'analyse du contexte donne le graphe de **figure 7.17** ci-dessous. Ce graphe de services et obtenues grâce notamment à l'application des règles de composition de services. Les services marqués en couleur verte sont des services d'acquisition de contexte qui sont initialement satisfaits tandis que les services en gris sont des services de traitement du contexte.

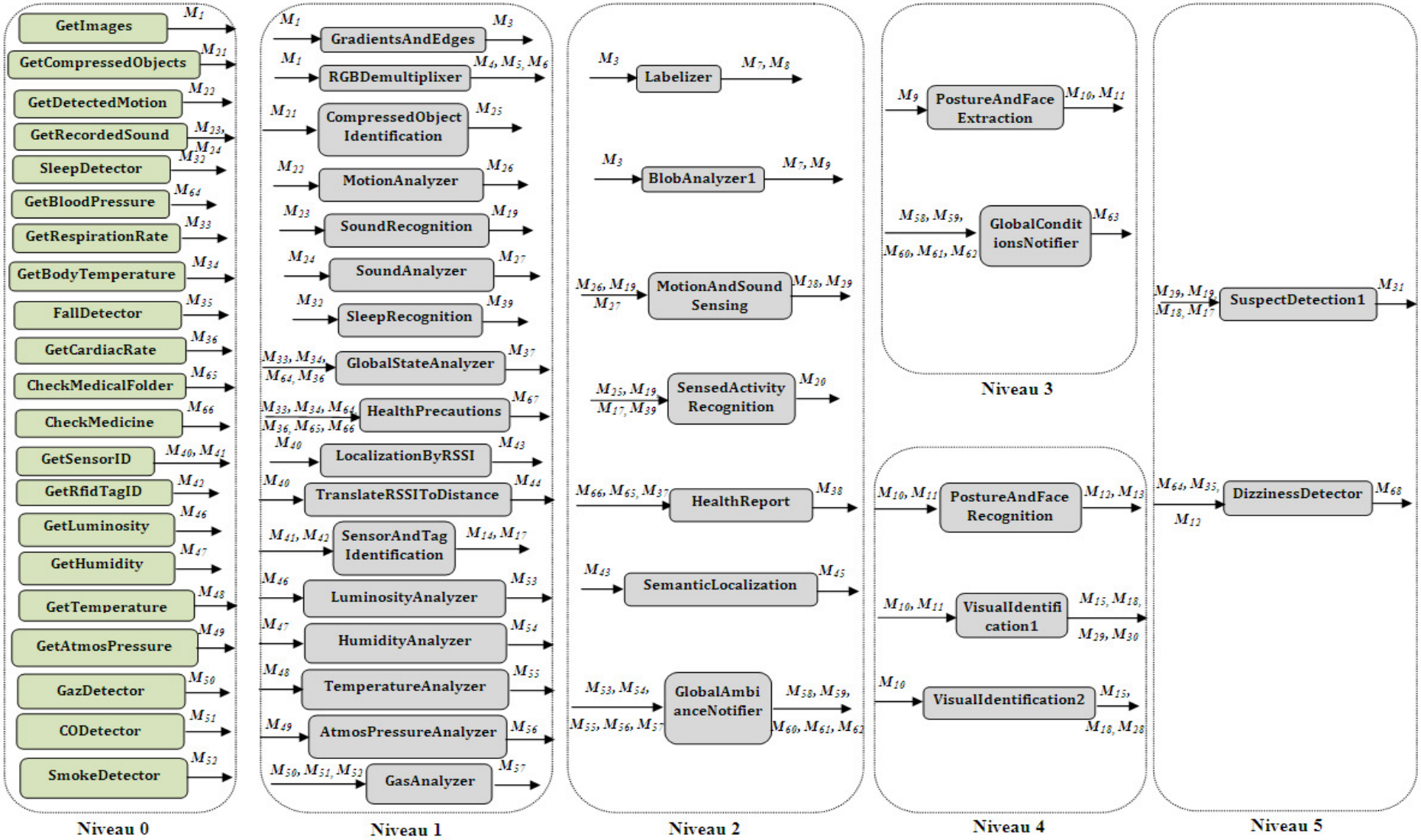


Figure 7.17 : Graphe global *AGoSC* des services abstraits disponibles.

7.3.2. Description des scenarii

En se basant sur les différents dispositifs et services disponibles, nous avons mis au point plusieurs scenarii afin de contrôler un espace intelligent et d'apporter les services nécessaires aux personnes qui s'y trouvent. Il s'agit notamment de personnes âgées à forte dépendance qui nécessitent une surveillance permanente et particulière. Dans ce contexte, l'environnement ambiant représente une maison intelligente dans laquelle sont déployés l'ensemble des dispositifs et services illustrés précédemment. Notre système de composition de services permet alors de gérer cet environnement d'une façon optimale et efficace. On admet que l'utilisateur de ce système est appelé *User*. Ce dernier se charge de la spécification des *règles événementielles* et c'est lui qui sera notifié en cas de la réalisation des objectifs associés à l'une de ces règles. Par ailleurs, plusieurs événements peuvent se produire dans cette maison intelligente impliquant ainsi différents scenarii qui correspondent à des configurations différentes des services disponibles. Certains scenarii consistent à reconnaitre l'activité d'une personne et détecter les mouvements suspects au niveau de la maison intelligente par la vidéo et le son, d'autres consistent à maintenir les conditions ambiantes dans cette maison, etc. Ci-après la description détaillée de cinq scenarii prototypiques.

7.3.2.1. Scenario1 : Reconnaissance d'activité

Ce scénario consiste à reconnaitre l'activité d'une personne en se basant sur des capteurs simples. Ce scénario est déclenché par le service sensible aux événements ***CompressedObjectsEvent*** qui contrôle les capteurs de pression installés sur les objets compressibles tels que les chaises, les fauteuils et les matelas. Ces capteurs envoient un signal dés qu'une pression est exercée sur l'objet en question. Le service ***CompressedObjectsEvent*** notifie à son tour le *module de détection d'événements* par l'occurrence d'un événement appelé «*ObjectIsCompressed* ». La *règle événementielle* associée à cette événement est appelée *ER1*. Cette règle est définie par l'utilisateur *User* de la manière suivante : *ER1*: *(User ; ObjectIsCompressed ; G1)* avec *G1 :({M20} ; {M20. Activity ≠ Null}).* Plusieurs services se mettent alors en collaboration afin d'atteindre l'objectif *G1*. Tous d'abords, le service ***GetCompressedObjects*** se charge de transmettre les identifiants de tous les objets compressés. Ces identifiants sont utilisés par le service ***CompressedObjectsIdentification*** afin d'associé un nom à l'objet compressé tels que chaise1, fauteuil2, etc. En parallèle à cette opération, le service ***SleepDetector*** détecte le sommeil des personnes grâce à des capteurs de sommeil installés sur ces derniers. Ces capteurs transmettent les somnolences détectées sur une personne. Le service ***SleepRecognition*** utilise cette information afin de donner l'état du sommeil de la personne (pas de sommeil, début de sommeil, sommeil léger, sommeil profond.). De même, les services ***GetSensorID*** et ***GetRfidTagID*** retournent tous les identifiants des objets mobiles détectés. Par la suite, le service ***SensorAndTagIdentification*** retourne les noms correspondants à ces identifiants. Un autre mode d'identification des personnes est aussi utilisé grâce à la reconnaissance vocale effectuée par le service ***SoundRecognition***. Ce dernier analyse le fichier audio transmis par le service ***GetRecordedSound*** afin de reconnaitre les personnes grâce à leur voix. Ce mode d'identification est très utile, notamment lorsque la personne ne porte ni de capteurs ni de tag RFID. Enfin, en couplant les différentes informations notamment l'identification des objets compressés, l'identification des personnes et leurs état de sommeil avec certaines

informations du contexte telles que la position des différents capteurs, l'heure d'acquisition et le comportement habituel des personnes identifiées, le service ***SensedActivityRecognition*** estime l'activité actuelle de la personne(par exemple, la personne A est assise sur le fauteuil2, la personne B est assise sur la chaise1 et elle est entrain de parler, etc.). Ainsi, le sous graphe *SubGoSC* extrait du graphe global *AGoSC* afin d'atteindre l'objectif de la règle *ER1* est illustré su la **figure 7.18**.

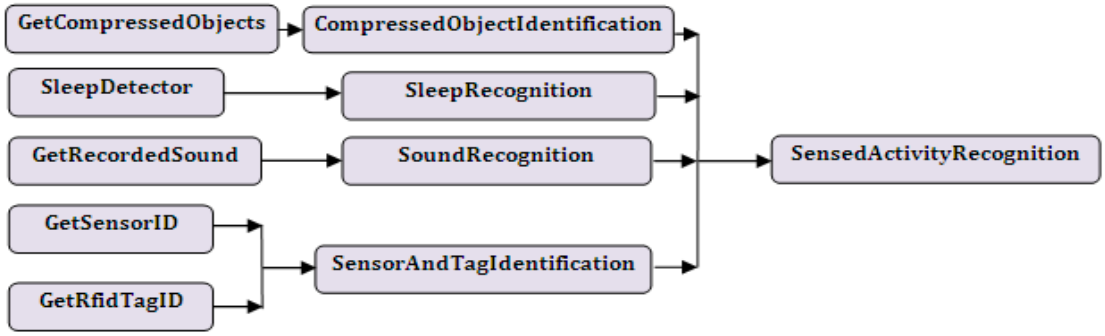


Figure 7.18 : Sous graphe *SubGoSC* du scénario « Reconnaissance d'activité ».

7.3.2.2. Scenario2 : Détection d'intrusions

Ce scénario consiste à confirmer l'existence d'une personne suspecte à l'intérieur de la maison. Ce scénario peut être déclenché par l'un des services sensibles aux événements suivant : ***MotionEvent*** et ***VisualEvent***. Le service ***MotionEvent*** contrôle les capteurs de mouvements installés à des endroits différents de la maison tandis que le service ***VisualEvent*** surveille des zones sensibles spécifiées sur le champ de vision d'une ou plusieurs caméras. Ainsi, lorsqu'un mouvement est détecté par les capteurs de mouvement ou dans une zone sensible spécifiée sur le champ de vision d'une caméra à une heure tardive de la nuit ou bien durant les périodes dans lesquelles le propriétaire de la maison est supposé être au travail, un événement appelé « *SuspectMotionIsOccured* » est aussitôt transmis au module *module de détection d'événements* par le service ***MotionEvent*** ou bien par le service ***VisualEvent***. La *règle événementielle* associée à cette événement est appelée *ER2*. Cette règle est définie par l'utilisateur *User* de la manière suivante : *ER2*: *(User ; SuspectMotionIsOccured ; G2)* avec *G2 :({M31} ; {M31.SuspectObjectAlert = True}).* Le processus de composition de services est alors déclenché afin d'atteindre l'objectif *G2*. Pour ce faire, tout d'abord, tous les mouvements enregistrés par le service ***GetMotion*** sont analysés par le service ***MotionAnalyzer***. Au même temps, tous le son enregistré par le service ***GetSound*** est analysés par le service ***SoundAnalyzer*** afin de vérifié s'il ya détection d'un bruit inhabituel. Le son est également utilisé par le service ***SoundRecognition*** afin d'effectuer une reconnaissance vocale des personnes présentes. Comme nous l'avons déjà souligné, ce mode d'identification est très utile, notamment lorsque la personne ne porte ni de capteurs ni de tag RFID. Par la suite, le son et les mouvements analysés sont corrélés et interprétés conjointement par le service ***MotionAndSoundSensing*** afin d'estimer le nombre d'objets voisins détectés et de vérifier s'il ya des personne reconnues par leurs voix. En parallèle à ces opérations, un scanne de tous les identifiants des capteurs et des tags présents est lancé par les deux services ***GetSensorID*** et ***GetRfidTagID*** afin de détecter les objets mobiles. Ces identifiants sont interprétés par le service ***SensorAndTagIdentification*** afin de trouver les personnes ou les objets

correspondants à ces identifiants. Dans cette composition, le mode d'identification visuel est également utilisé afin d'augmenter la probabilité de reconnaitre les personnes présentes. Pour ce faire, Pour ce faire, il s'agit d'abord d'activer la transmission des flux vidéo depuis les caméras contrôlées par le service ***GetImages***. Ce dernier renvoie une séquence d'images (frames) JPEG contenant tous les objets détectés dans les champs de vision des caméras. Ces images sont brutes et riches en termes d'information mais elles nécessitent un prétraitement afin de se focaliser uniquement sur les parties contenant la personne afin de faciliter son identification. Ce prétraitement est réalisé, en premier lieu, par le service ***GradientsAndEdges*** qui renvoie des images simple en noir et blanc. Ce service conservent uniquement les contours des objets en leurs attribuant des pixels blanc et le reste de l'image étant en noir. En seconde lieu, les contours des différents objets sont identifiés comme étant des blob physiques sur l'image, puis encadrés dans des rectangles par le service ***BlobAnalyzer1***. Ce service renvoie une séquence d'image contenant chacune un blob physique. Parmi ces blobs, le service ***PostureAndFaceExtraction*** ne garde que les blobs ressemblant à une posture ou bien une face d'une personne. En troisième lieu, le service ***VisualIdentification1*** utilise la correspondance de l'allure des postures et des faces détectées avec celles enregistrées dans une base de donnée pour chaque personnes afin d'identifier la personne en question. Enfin, en couplant toutes les informations relatives aux différents modes d'identification avec certaines informations du contexte telles que l'heure et la position de détection ainsi que l'agenda de l'utilisateur (vérifier, par exemple, que rien n'est prévu pour cette heure-ci), le service ***SuspectDetection1*** estime le niveau d'alerte sur la détection d'intrusion. Ainsi, le sous graphe *SubGoSC* extrait du graphe global *AGoSC* afin d'atteindre l'objectif de la règle *ER2* est illustré su la **figure 7.19**.

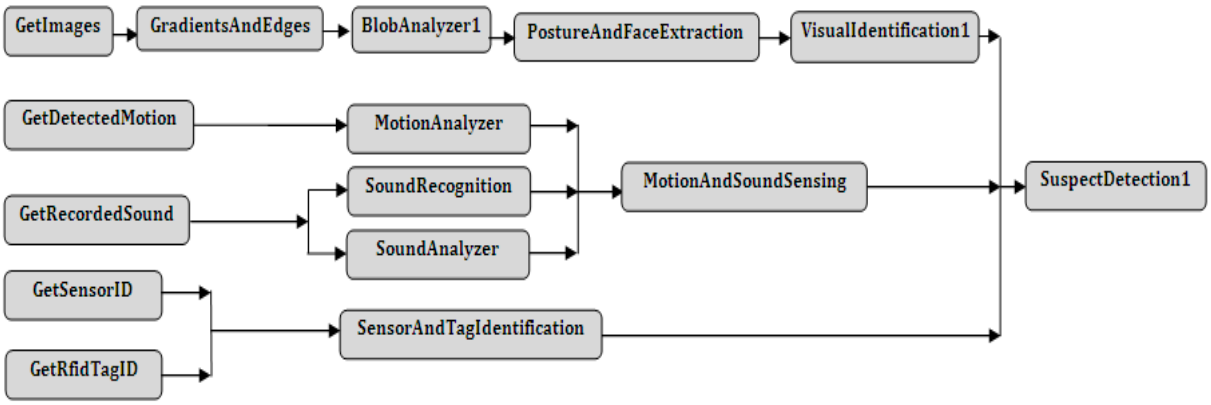


Figure 7.19 : Sous graphe *SubGoSC* du scénario « Détection d'intrusion ».

7.3.2.3. Scenario3 : Étourdissements après un lever brutal

Nombre d'étourdissements sont dus à l'hypotension orthostatique. Cette appellation compliquée désigne une baisse ponctuelle de la pression artérielle lors du passage brusque de la posture assise ou couchée à la position debout. Cette hypotension brutale peut entraîner des malaises, des vertiges, voire une perte de connaissance. Cette situation peut être déclenchée par le service sensible aux événements ***BloodEvent*** qui permet de détecter une baisse anormale de la pression artérielle. Lorsqu'un tel événement est détecté, le service ***BloodEvent*** envoie aussitôt un événement appelé « *AbnormalBoodPressure* » au *module de détection d'événements.* La *règle événementielle* associée à cette événement est appelée *ER3*. Cette

règle est définie par l'utilisateur *User* de la manière suivante : *ER3*: *(User ; AbnormalBoodPressure ; G3)* avec *G3 :({M68} ; {M68. Etourdissement ∈ {Malaise, Vertige, Perte de connaissances}}).* L'objectif *G3* peut être atteint par une composition des services de reconnaissance de postures, de mesure de la pression artérielle et de détection de chutes. Pour ce faire, la portion du sous graphe du scénario 2 (**Figure 7.19)** située entre le service ***GetImages*** et le service ***PostureAndFaceExtraction*** est réutilisée dans ce présent scénario afin de reconnaitre la posture d'une personne par le service ***PostureAndFaceRecognition***. Ce dernier estime la posture d'une personne (assise, debout, allongée, etc.) ainsi que l'humeur de son visage (content, énervé, etc.). Une situation d'étourdissement est détectée par le service ***DizzinessDetector*** en analysant conjointement plusieurs postures successives de la personne et les pressions artérielles correspondantes à chaque passage entre deux postures. Le service de détection de chutes ***FallDetector*** est aussi utilisé afin de confirmer l'étourdissement. Par exemple, si une chute est détectée et la posture de la personne est stabilisée à la valeur « allongée » pendant plusieurs minutes, le service ***DizzinessDetector*** peut inférer qu'il s'agit probablement d'une perte de connaissance. Ainsi la personne doit absolument être prise en charge et soignée car cet état d'évanouissement peut occasionner aussi des blessures graves. Ainsi, le sous graphe *SubGoSC* extrait du graphe global *AGoSC* afin d'atteindre l'objectif de la règle *ER3* est illustré su la **figure 7.20**.

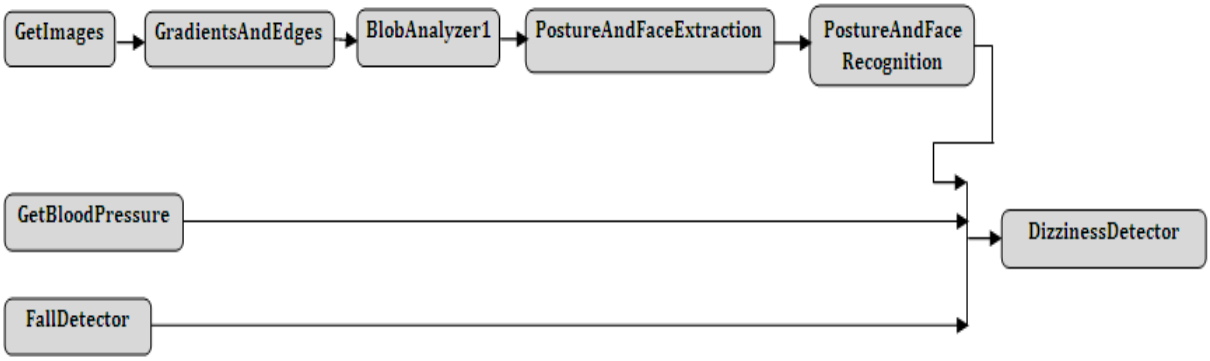


Figure 7.20 : Sous graphe *SubGoSC* du scénario « Étourdissements après un lever brutal ».

7.3.2.4. Scenario4 : Edition d'un bilan de santé global

Ce scénario consiste à surveiller l'état de santé global d'une personne et le rappeler à prendre certaines précautions en cas de besoin. Ce scénario peut être déclenché périodiquement, par exemple en début et en fin de journée, par le service sensible aux événements ***HealthStateEvent***. Lorsque l'heure prévue pour l'analyse de l'état de santé global de la personne est arrivée, le service ***HealthStateEvent*** envoie aussitôt un événement appelé « *IsTimeForHealthState* » au *module de détection d'événements.* La *règle événementielle* associée à cette événement est appelée *ER4*. Cette règle est définie par l'utilisateur *User* de la manière suivante : *ER4 (User ; IsTimeForHealthState ; G4)* avec *G4 :({M38, M67} ; {M38.HealthAlert ∈ {2,3}}).* Afin d'atteindre l'objectif *G4* par une composition des services, tout d'abord, les quatre paramètres de santé suivants : rythme cardiaque, rythme respiratoire, la température corporelle et la pression artérielle sont transmis respectivement par les services: ***GetCardiacRate***, ***GetResperationRate***, ***GetBodyTemperature*** et ***GetBloodPressure***. Ensuite, ces différents paramètres sont analysés conjointement par le service ***GlobalStateAnalyser*** pour vérifier s'il ya une quelconque anomalie. En utilisant ces

paramètres, le service ***HealthPrecautions*** propose aussi certaines précautions à la personne pour diminuer les risques de complication en se basant sur son dossier médicale transmis par le service ***CheckMedicalFolder*** et ses différentes prises de médicaments transmises par le service ***CheckMedicine***. Par exemple, si les rythmes respiratoire et cardiaque sont augmentés, ce service propose à la personne de diminuer l'activité physique, de même si la température corporelle augmente, le service ***HealthPrecautions*** suggère de boire de l'eau ou de prendre un bain. Ce service peut également rappeler la personne s'elle a oublié de prendre son médicament. En parallèle à cette opération de sensibilisation de la personne, le service ***HealthReport*** estime le niveau d'alerte sur l'état de santé de la personne en se basant sur l'analyse globale qui a été effectuée ainsi que le dossier médicale de la personne transmis par le service ***CheckMedicalFolder*** et les différentes prises de médicaments transmises par le service ***CheckMedicine***. Ainsi, le sous graphe *SubGoSC* extrait du graphe global *AGoSC* afin d'atteindre l'objectif de la règle *ER4* est illustré su la **figure 7.21**.

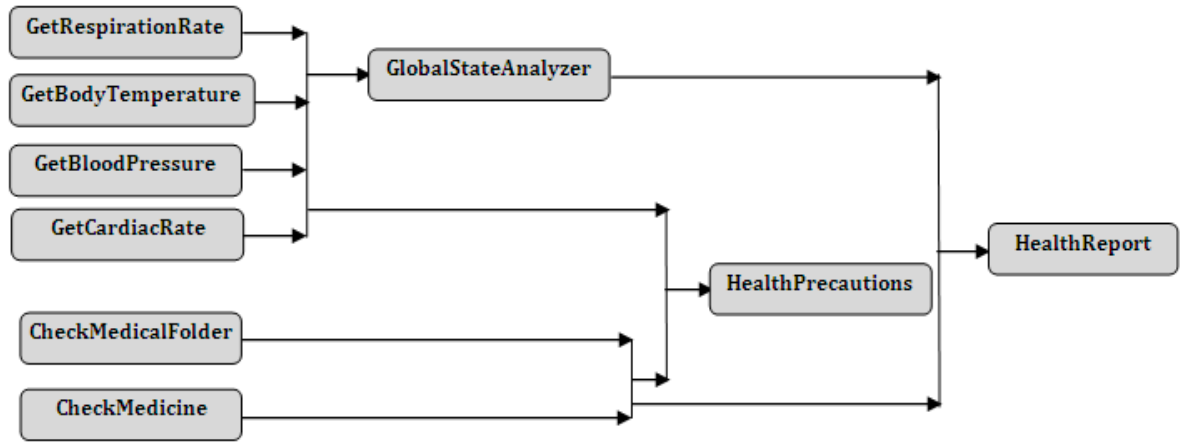


Figure 7.21 : Sous graphe *SubGoSC* du scénario « Édition d'un bilan de santé global ».

7.3.2.5. Scenario5 : Détection des conditions ambiantes anormales

Ce dernier scenario consiste à surveiller les conditions ambiantes qui règnent au sein de la maison intelligente et à envoyer une alerte en cas d'une quelconque anomalie. Ce scénario peut être déclenché par l'un des services sensibles aux événements suivant : ***GasEvent*** et ***AmbienceEvent***. Le service ***GasEvent*** contrôle les capteurs de fumée et de monoxyde de carbone tandis que le service ***AmbienceEvent*** contrôle les capteurs de température et d'humidité. Ainsi, lorsqu'un gaz est détecté par les capteurs de gaz ou bien la température ou l'humidité dépassent les seuils spécifiés au préalable, un événement appelé « *AbnormalConditionIsOccured* » est aussitôt transmis au module *module de détection d'événements* par le service ***GasEvent*** ou bien par le service ***AmbienceEvent***. La *règle événementielle* associée à cette événement est appelée *ER5*. Cette règle est définie par l'utilisateur *User* de la manière suivante : *ER5*: *(User ; AbnormalConditionIsOccured ; G5)* avec *G5 :({M63} ; {M63.GlobalAlert ∈ {1, 2, 3}}).* Afin d'atteindre l'objectif *G5* par une composition des services, il s'agit d'abord d'analyser les paramètres ambiants suivants : luminosité, humidité, température et pression atmosphérique échantillonnés respectivement par les services suivants : ***GetLuminosity***, ***GetHumidity***, ***GetTemperature*** et ***GetAtmosPression***. La qualité de l'air à l'intérieur est aussi analysée par le service ***GasAnalyzer*** qui se base sur les services détecteurs de gaz, de CO et de fumée. Par la suite,

les différentes analyses effectuées sont transmises au service ***GlobalAmbianceNotifier*** qui se charge d'attribuer un niveau d'alerte à chaque analyse. Enfin, à la lumière de ces différentes alertes transmises, le service ***GlobalConditionsNotifier*** édite un niveau d'alerte global sur les conditions environnementales qui règnent au sein de l'espace intelligent. Ainsi, le sous graphe *SubGoSC* extrait du graphe global *AGoSC* afin d'atteindre l'objectif de la règle *ER5* est illustré su la **figure 7.22**.

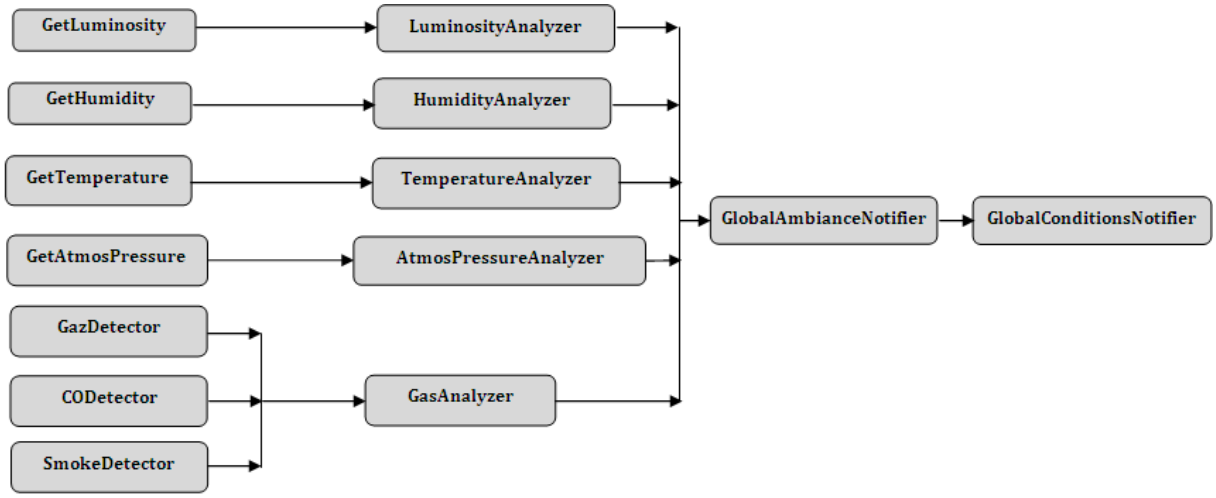
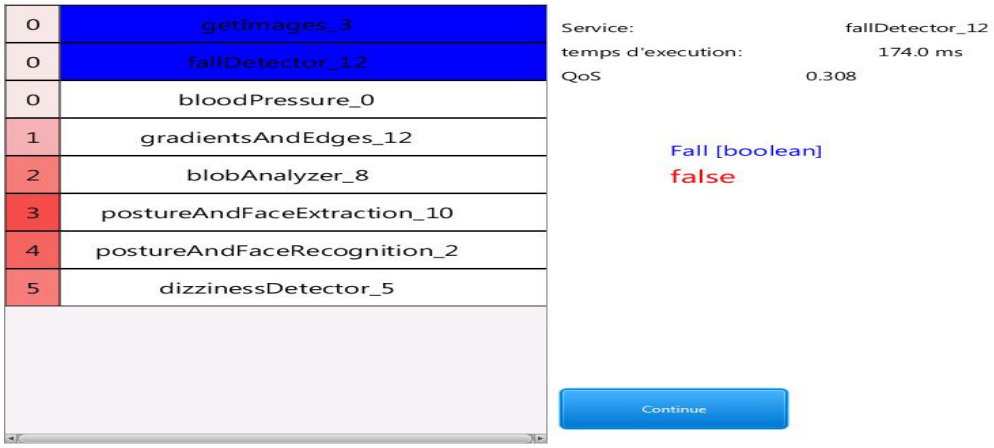


Figure 7.22 : Sous graphe *SubGoSC* du scénario «Détection des conditions anormales ».

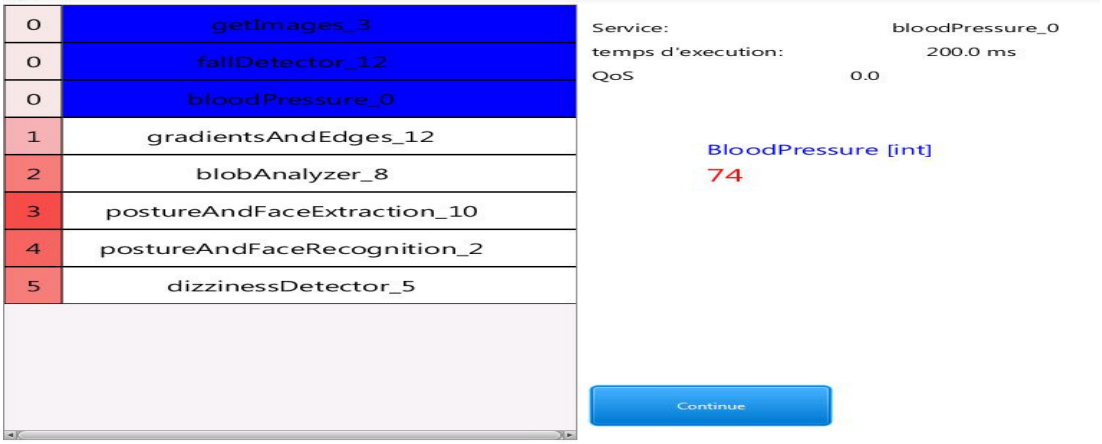
7.3.3. Exécution du scénario 3 : « Étourdissements après un lever brutal »

L'interface graphique montrée dans **la section 2.2.5** permet de visualiser l'exécution automatique des différents scénarios illustrés dans les sections précédentes. Nous pouvons choisir le mode d'exécution pas-à-pas afin de voir toutes les étapes intermédiaires d'exécution d'un scénario. Dans cette section, nous avons choisi de montrer l'exécution du scénario 3 qui vérifie l'existence d'étourdissements chez une personne. La liste des services concrets sélectionnés pour exécuter ce scénario ainsi que leur niveau respectif dans le sous graphe *SubGoSC* sont illustrés sur la partie gauche de la **figure 7.23** tandis que le résultat d'exécution du service en cours est montré sur la partie droite de cette même figure. Cette figure montre également le temps d'exécution effectif du dernier service exécuté ainsi que sa qualité de service fournie. Les services concrets déjà exécutés sont tous marqués en bleu. Le processus d'exécution est détaillé dans les figures suivantes :

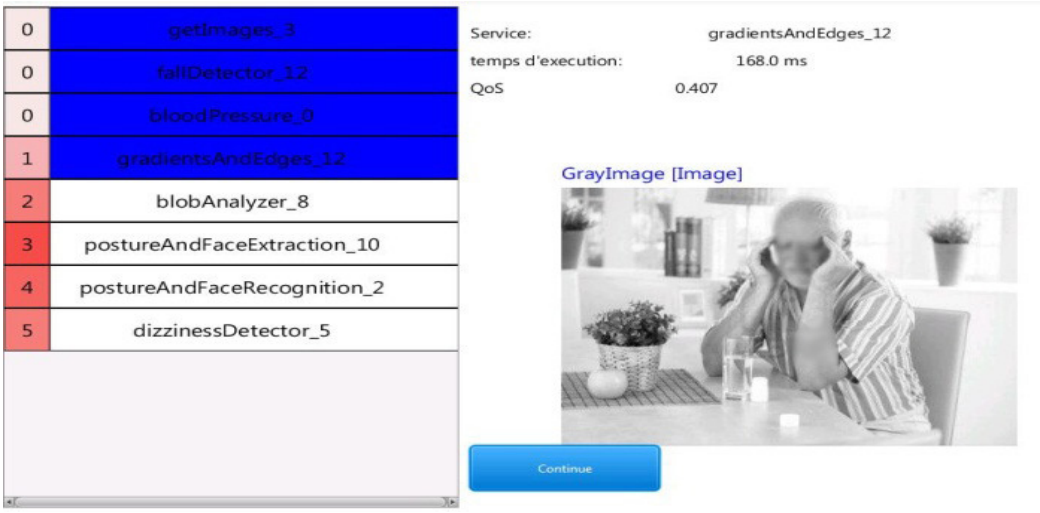
(a) Résultat de l'exécution du service *getImages_3* du niveau 0 du graphe *SubGoSC*.



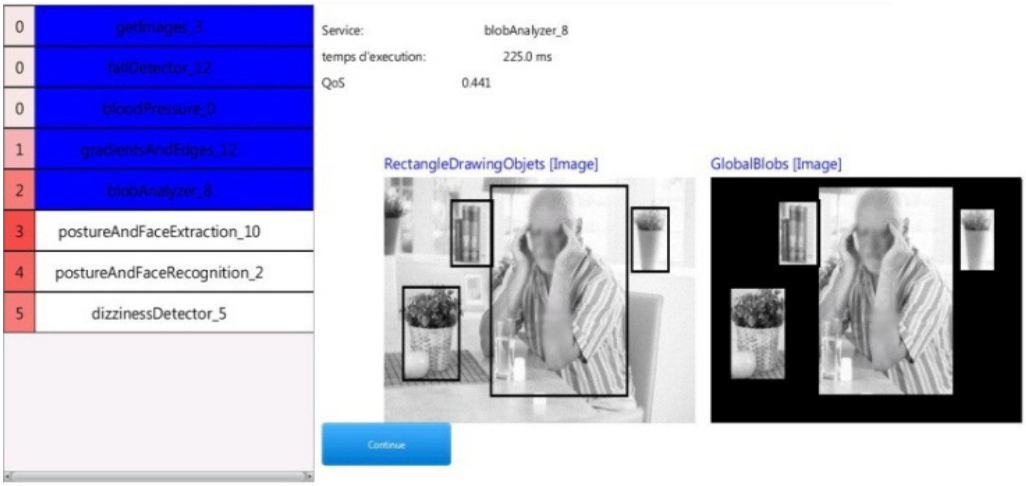
(b) Résultat de l'exécution du service *fallDetector_12* du niveau 0 du graphe *SubGoSC*.



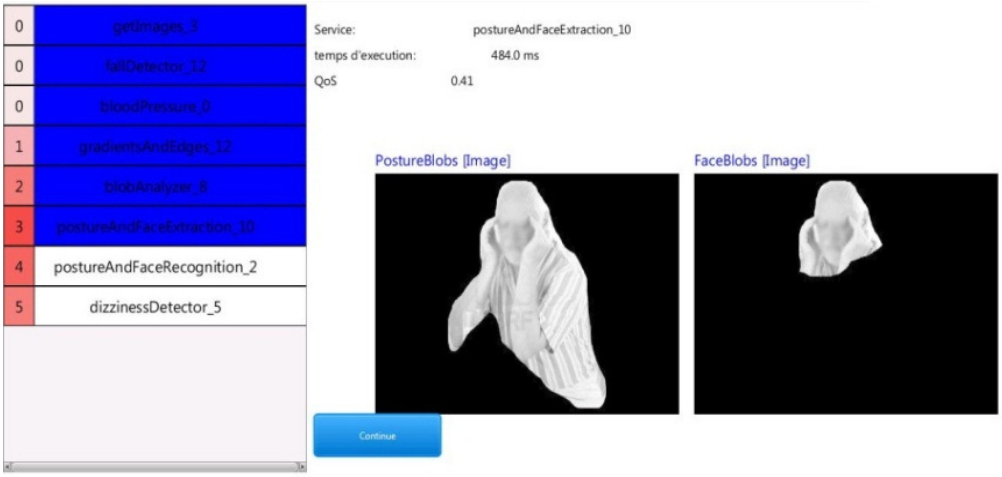
(c) Résultat de l'exécution du service *bloodPressure_0* du niveau0 du graphe *SubGoSC*.



(d) Résultat de l'exécution du service *gradientsAndEdges*_12 du *niveau1* du graphe *SubGoSC*.

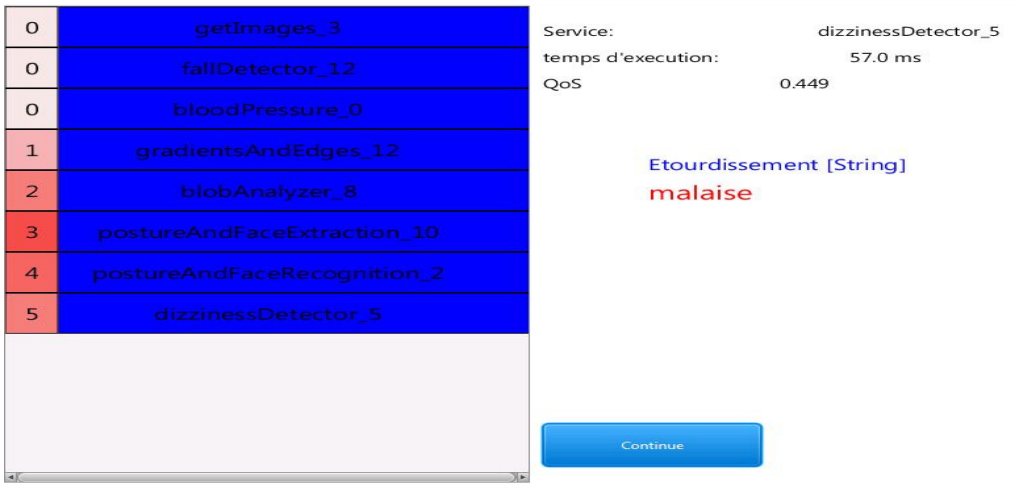


(e) Résultat de l'exécution du service *blobAnalyzer_8* du *niveau2* du graphe *SubGoSC.*



(f) Résultat de l'exécution du service *postureAndFaceExtraction_10* du *niveau 3* du graphe *SubGoSC.*

(g) Résultat de l'exécution du service *postureAndFaceRecognition_2* du *niveau 4* du graphe *SubGoSC.*



(h) Résultat de l'exécution du service *dizzinessDetector_5* du *niveau 5* du graphe *SubGoSC*.

Figure 7.23 : Exécution détaillée du scénario « Étourdissements après un lever brutal ».

Le résultat final de l'exécution du scénario 3 montre que la personne est vraisemblablement male à l'aise. Ce malaise doit être pris en charge le plus rapidement possible afin d'éviter tout risque de complication. Dans ce genre de situation, l'utilisateur *User* spécifie un plan d'actions pour établir un lien direct avec la personne concernée. Ce plan d'action est indiqué préalablement au système pour qu'il soit exécuté en cas ou l'objectif associé à la règle *ER3* du scénario 3 est atteint. Etant donné que l'utilisateur souhaite établir un contact avec la personne afin de l'assister et de vérifier de prés son état de santé, un plan d'action adéquat peut être spécifié comme suit :

- Notifier l'utilisateur *User ;*
- Activer la localisation de la personne par Cricket ;
- Déplacement du robot vers la personne ;
- Activer la caméra du robot ;
- Etablir un contacte interactif via *Skype* entre la personne et l'utilisateur *User*.

Ce plan d'actions peut être réalisé par une composition de services comme le montre la **figure 7.24.**

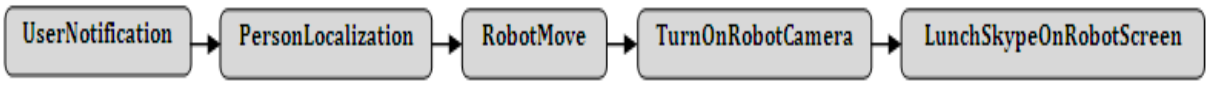


Figure 7.24 : Plan d'actions après un étourdissement

Une fois ce plan est exécuté, le robot *Kompai* se trouve en face de la personne et un lien sera établi entre l'utilisateur *User* et la personne malade via une connexion *Skype* comme le montre la **figure 7.25**.

Figure 7.25 : Contact direct entre la personne malade et l'utilisateur *User* via *Kompai*

7.3.4. Monitoring du scénario 1 : « Reconnaissance d'activité »

L'objectif de cette section est de montrer les différents cas de monitoring afin d'assurer la continuité de services en cas de pannes. Pour ce faire, nous avons choisi d'illustrer le monitoring du scénario1 qui porte sur la reconnaissance d'activité d'une personne. Le sous graphe *SubGoSC* associé à ce scénario est montré sur la **figure 7.18** de la **section 7.3.2.1.** Le monitoring de service est envisageable dans les trois cas de figure suivants : panne d'un service concret, panne d'un service abstrait, et coupure d'un chemin vers l'objectif. Ces trois cas sont cités dans l'ordre croissant du moins critique au plus critique.

7.3.4.1. Cas1 : panne d'un service concret

Le monitoring dans ce cas de figure se base sur un mécanisme de substitution local consiste à remplacer le service concret en panne par un autre service concret du même service abstrait. Etant donné que tous les services concrets appartenant à un même service abstrait sont ordonnés selon leur qualité de service, alors il est très facile de remplace un service concret qui tombe en panne par un autre service concret qui fourni une meilleur qualité de service. Ce remplacement est effectué en exécutant une nouvelle sélection de services. Supposant que le service concret *SensorAndTagIdentification_3* est initialement sélectionné pour le service abstrait *SensorAndTagIdentification*. Le service *SensorAndTagIdentification_3* est embarqué sur le robot *Kompai* et il est accessible via une liaison Wifi. Supposant que lors de l'invocation de ce service, la connexion Wifi avec le robot est temporairement interrompue. Dans ce cas, Le service *SensorAndTagIdentification_3* est immédiatement remplacé par le service *SensorAndTagIdentification_5* qui fourni la meilleure qualité. Ce service se trouve sur un PC portable muni d'une connexion filaire.

7.3.4.2. Cas 2 : panne d'un service abstrait

Le monitoring dans ce cas de figure se base également sur un mécanisme de substitution local, mais cette fois-ci, il consiste à remplacer tout un service abstrait par un autre service abstrait fournissant certaines fonctionnalités similaires. Un service abstrait est considéré comme étant en panne lorsque l'exécution de tous ses services concrets est échouée pour une raison ou une autre. Cette panne peut être détectée par le mécanisme de monitoring appliqué au cas précédent. Le remplacement d'un service abstrait tombé en panne s'effectue au même niveau du graphe *SubGoSC* où cette panne s'est produite. Le nouveau service abstrait remplaçant doit être capable de fournir les messages de sortie que le service remplacé n'a pas pus fournir dans le graphe *SubGoSC*. Pour illustrer ce cas de figure, supposant que tous les

services concrets appartenant au service abstrait *SensorAndTagIdentification* n'arrivent pas à s'exécuter correctement. On constate que le service abstrait *SensorIdentification* est satisfait au niveau 1 du graphe *SubGoSC* (**Figure 7.18** de la **section 7.3.2.1**) et permet de fournir les deux messages *M14* et *M17* que le service *SensorAndTagIdentification* n'a pas pus fournir. Donc, le service *SensorAndTagIdentification* est immédiatement remplacé par le service *SensorIdentification* afin de remplir les fonctionnalités désirées. Admettant que même le service abstrait *SensorIdentification* tombe en panne, alors il sera remplacé à son tour par le service abstrait *RfidTagIdentification* pour les mêmes raisons.

7.3.4.3. Cas3 : coupure d'un chemin vers l'objectif

Lorsqu'aucun service abstrait n'est trouvé pour remplacer le service en panne, le chemin menant vers l'objectif passant par ce service sera complètement coupé. Il est alors nécessaire d'établir un nouveau chemin menant vers l'objectif désiré. Dans ce cas de figure, le monitoring se base sur un mécanisme de substitution global qui consiste à faire une recomposition (re-planification) de services afin d'établir un chemin alternatif vers l'objectif. Pour illustrer ce cas de figure, admettant que les trois services abstrait *SensorAndTagIdentification, SensorIdentification et RfidTagIdentification* tombent tous en panne. Dans ce cas, le résultat de la recomposition de services est complètement un nouveau scenario qui consiste à reconnaitre l'activité d'une personne en se basant uniquement sur la reconnaissance visuelle. Pour ce faire, il s'agit d'abord d'activer la transmission des flux vidéo depuis les caméras contrôlées par le service ***GetImages***. Ce dernier renvoie une séquence d'images (frames) JPEG contenant tous les objets détectés dans les champs de vision des caméras. Ces images sont brutes et riches en termes d'information mais elles nécessitent un prétraitement afin de se focaliser uniquement sur les parties contenant la personne afin de faciliter son identification. Ce prétraitement est réalisé, en premier lieu, par le service ***GradientsAndEdges*** qui renvoie des images simple en noir et blanc. Ce service conservent uniquement les contours des objets en leurs attribuant des pixels blanc et le reste de l'image étant en noir. En seconde lieu, les contours des différents objets sont identifiés comme étant des blob physiques sur l'image, puis encadrés dans des rectangles par le service ***BlobAnalyzer1***. Ce service renvoie une séquence d'image contenant chacune un blob physique. Parmi ces blobs, le service ***PostureAndFaceExtraction*** ne garde que les blobs ressemblant à une posture ou bien une face d'une personne. En troisième lieu, le service ***VisualIdentification1*** utilise la correspondance de l'allure des postures et des faces détectées avec celles enregistrées dans une base de donnée pour chaque personnes afin d'identifier la personne en question. En parallèle, le service ***PostureAndFaceRecognition*** estime la posture de la personne (assise, debout, allongée, etc.) ainsi que l'humeur de son visage (content, énervé, etc.). Enfin, en couplant les informations envoyées par les deux services précédents avec certaines informations du contexte telles que la position des caméras, l'heure d'acquisition et le comportement habituel de la personne identifiée, le service ***VisualActivityRecognition2*** estime l'activité actuelle de la personne (prépare le diner, se prépare pour coucher, assise sur un fauteuil, etc.). Ainsi, le nouveau sous graphe *SubGoSC* qui permet d'atteindre l'objectif de la règle *ER1* est illustré su la **figure 7.26**.

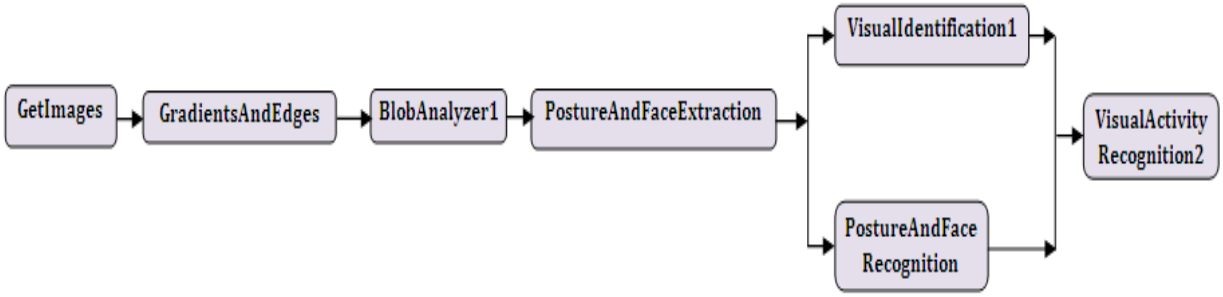


Figure 7.26 : Sous graphe de reconnaissance d'activité en mode visuel.

7.4. Tests et évaluation des performances

7.4.1. Caractéristiques matérielles et logicielles

Afin de montrer la faisabilité du Framework *FrEvASeC* et d'évaluer concrètement ses performances, nous avons implémenté tous les algorithmes de ce Framework en Java en utilisant la version 1.6.0_21 sous Eclipse IDE. Tous les tests d'évaluations empiriques ont été réalisés sur un PC ayant les caractéristiques suivantes : un processeur Pentium 4 à 1,6 GHz, une RAM de taille 1Go, et un système d'exploitation Windows XP. Dans ce travail, nous avons mené quatre tests différents pour montrer l'impact des différents paramètres préconfigurés sur les performances du Framework *FrEvASeC*. Le premier et le deuxième test étudient l'impact de la variation, respectivement, du nombre de services abstraits et du nombre des événements survenus sur le temps de réponse de l'algorithme de construction du graphe *AGoSC* (*AGoSC Construction Algorithm*) et de l'algorithme d'extraction du sous graphe *SubGoSC* (*SubGoSC Extraction Algorithm*). Le troisième et le quatrième test étudient l'impact de la variation, respectivement, du nombre de services concrets dans chaque service abstrait et du seuil apprentissage sur les performances de l'algorithme de monitoring de services (*Services Monitoring Algorithm*).

7.4.2. Impact de la variation des services abstraits sur le temps de réponse

Dans cette évaluation, nous avons empiriquement mesuré l'évolution du temps de réponse de l'algorithme de construction du graphe *AGoSC* (*AGoSC Construction Algorithm*) et de l'algorithme d'extraction du sous graphe *SubGoSC* (*SubGoSC Extraction Algorithm*) en fonction du nombre de services abstraits disponibles. Pour ce faire, nous avons varié le nombre de services abstrait disponibles de 10^2 à 10^3 par un pas de 10^2 services générés aléatoirement (**Figure 27**). Le nombre max de messages pour chaque service généré est égal à 5 pour les messages d'entrée et 5 pour les messages de sortie. Ce nombre représente, en moyenne, le nombre de messages consommés ou produits par un service réel. Les résultats obtenus de cette évaluation sont comparés aux trois approches suivantes: ***FCoSC [165, 166], CDSC [167, 168] et HTN-DL [164]***. La notion de tâches utilisées dans ces approches correspond à la notion d'événements utilisés dans ce travail. Le premier test est effectué en fixant le nombre d'événements à 5 (**Figure 7.27.a**) tandis que le deuxième test est effectué en fixant le nombre d'événements à 10 (**Figure 7.27.b**).

La **figure 7.27** montre clairement que le temps de réponse total pris par l'algorithme de construction du graphe *AGoSC* et l'algorithme d'extraction du sous graphe *SubGoSC* est

nettement meilleur à ceux des algorithmes *FCoSC, CDSC, et HTN-DL*. En outre, cette figure montre que le temps nécessaire pour la construction du graphe *AGoSC* reste constant même lorsque le nombre de tâches (i.e. événements survenus) augmente. Cela s'explique par le fait que l'algorithme de construction du graphe *AGoSC* est complètement indépendant du nombre d'événements survenus. Cet algorithme ne dépend que du nombre de services abstraits disponibles. Cependant, l'algorithme d'extraction du sous graphe *SubGoSC* est étroitement lié au nombre d'événements qui se produisent. En effet, l'extraction d'un sous graphe est effectuée uniquement lorsqu'un événement se produit dans l'environnement. Ainsi, le temps d'extraction des sous graphes augmente avec l'augmentation du nombre d'événements comme le montrent les **figures 7.27.a et 7.27.b**. Toutefois, cette augmentation reste toujours meilleure par rapport aux approches *FCoSC, CDSC, et HTN-DL.*

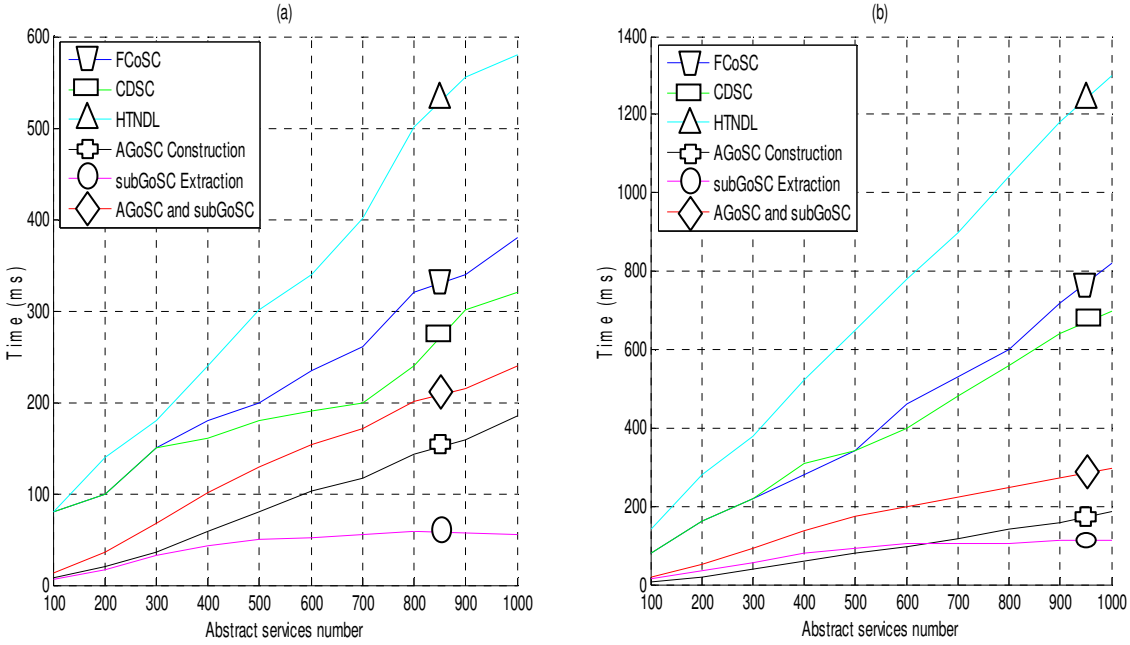


Figure 7.27 : Temps de composition de services vs. Nombre de services abstraits: (a) nombre de tâches (i.e. événements) = 5, (b) nombre de tâche = 10.

La **figure 7.28** montre le passage à l'échelle de l'algorithme de construction du graphe *AGoSC*. Nous constatons que, pour un nombre de services abstraits relativement petit, la construction du graphe *AGoSC* s'effectue très rapidement. Cependant, lorsque le nombre de services abstraits déployés devient assez grand, le temps de construction du graphe *AGoSC* peut devenir très long, car la construction, dans ce cas, procède à l'exploration et l'identification des services appropriés parmi un grand nombre de services disponibles. Malgré cette lenteur enregistrée, les résultats de la **figure** 7.**28** montrent une augmentation d'une manière tout à fait acceptable du temps de construction du graphe *AGoSC*. Les **figures** 7.**28.a et 7.28.d** montrent que le temps de construction augmente d'une façon quasi-linéaire, tandis que les **figures 7.28.b et 7.28.c** montrent quelques fluctuations au niveau de ce temps de construction. En effet, le temps de construction dépend également du nombre de messages d'entrées/sorties des services abstraits, car les règles de composition de services se basent sur l'exploration de ces messages. Ainsi, les fluctuations enregistrées sont dues au fait que

certains services abstraits générés possèdent moins de messages d'entrées/sorties. Donc, l'application des règles de composition de services devient plus rapide, d'où une diminution du temps de construction qui génère une fluctuation. D'une façon générale, si on a deux ensembles A et B de services abstraits telle que la taille de A et supérieure à celle de B, alors le temps de construction dans l'ensemble A n'est pas forcement supérieur à celui dans B car ce temps dépend aussi du nombre de messages d'entrées/sorties des services contenus dans A et B. En tout état de cause, les résultats obtenus montrent que l'implémentation actuelle permet de construire un graphe global de tous les services abstraits disponibles dans un délai tout à fait raisonnable (moins de 20s pour 10^5 services abstraits). Ce temps est raisonnable parce que la construction du graphe *AGoSC* est réalisée en phase offline (c'est à dire avant la détection de tout événement). Ainsi, ce temps n'affecte pas la réactivité du Framework *FrEvASeC* à un événement qui survient dans l'environnement (en online).

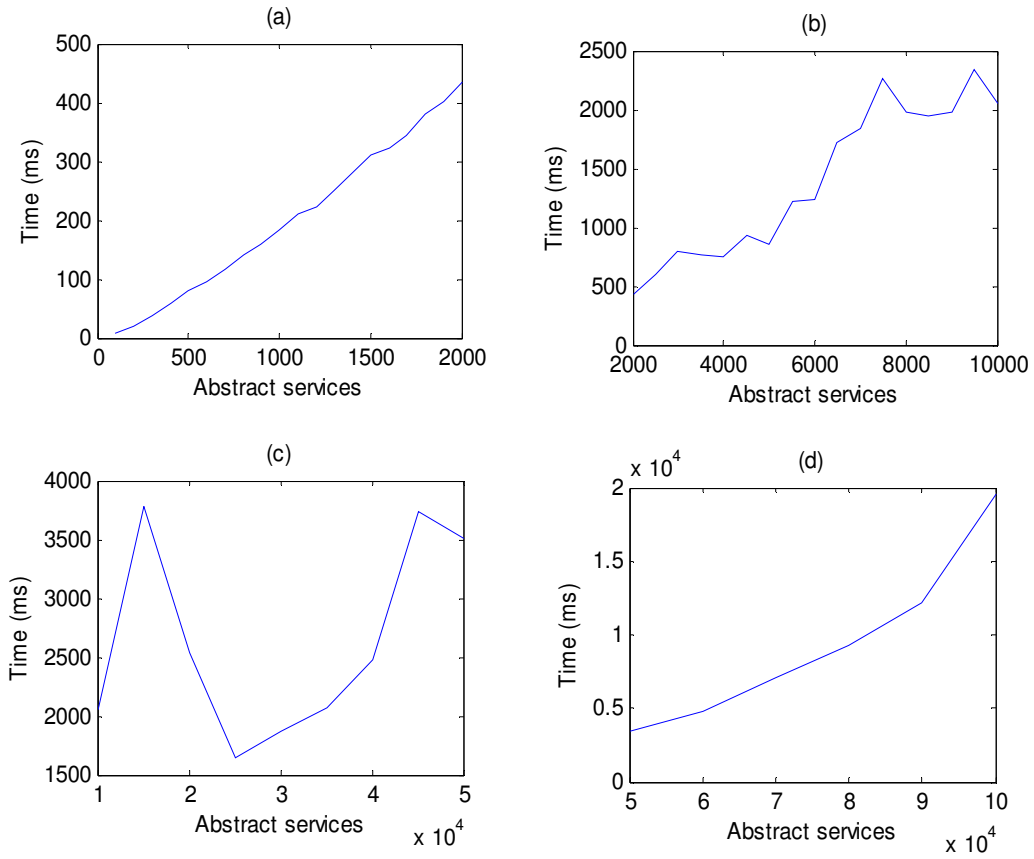
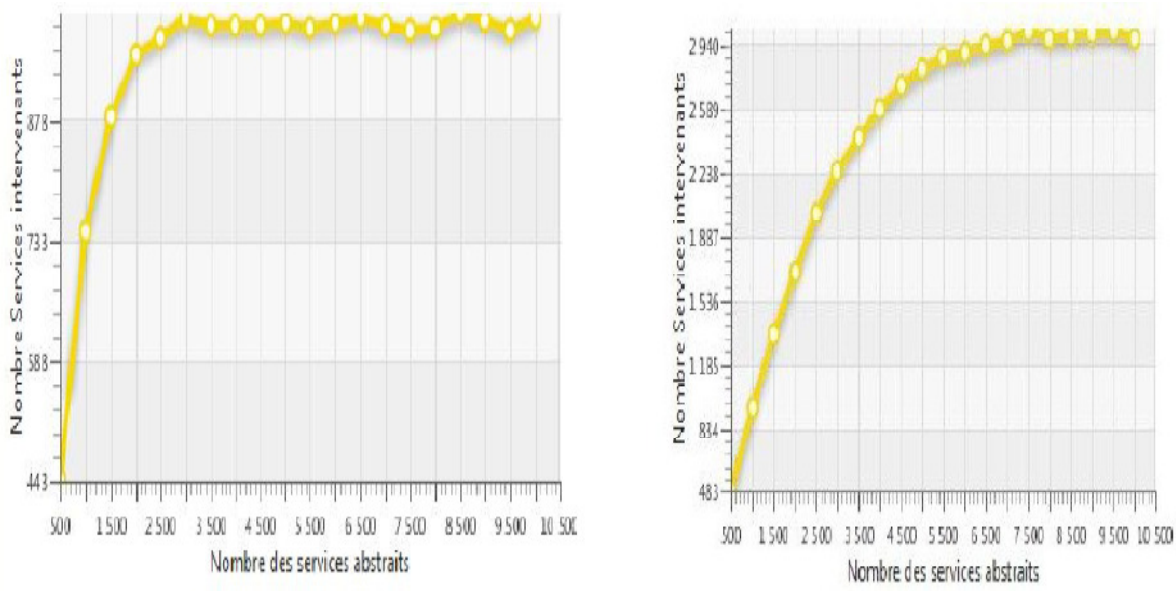


Figure 7.28 : Scalabilité de l'algorithme de construction du graphe *AGoSC*

Afin de mieux connaitre le nombre effectif de services abstraits qui participent réellement à la composition du graphe global *AGoSC*, nous avons mesuré la taille de ce graphe en fonction du nombre de services disponibles. Pour ce faire, nous avons mené deux évaluations différentes. Dans la première évaluation, nous avons fixé le nombre total de messages à 2000 et nous avons varié le nombre de services abstraits disponibles de 500 et 10^4 par un pas de 500 services (**Figure 7.29.a**). La deuxième évaluation garde les mêmes variations des services abstraits que la précédente, sauf que, le nombre de messages, cette fois-ci, est égal à 6000

(**Figure 7.29.b**). Il faut noter que les services abstraits générés utilisent des messages d'entrée/sortie parmi l'ensemble des messages initialement fixés. Les résultats de ces deux évaluations sont montrés sur les graphes de la **figure 7.29**.



(a) (b)

Figure 7.29 : Taille du graphe *AGoSC* en fonction du nombre de services et de messages

Nous constatons que le nombre de services intervenants (i.e. taille du graphe global) lors de la construction du graphe global augmente proportionnellement avec le nombre de services abstraits existants dans l'environnement jusqu'à une certaine limite, où le nombre des services intervenants se stabilise. Ce seuil est d'environ 1000 services pour la **figure 7.29.a**, et il est d'environ 2900 pour la **figure 7.29.b**. Cette stabilité est due au nombre total limité des messages d'entrée/sortie. En effet, la stabilisation de la taille du graphe global est influencée par le nombre de messages disponibles. Nous expliquons cette stabilité par le fait que, après l'exploration de tous les messages existants, l'augmentation du nombre de services abstraits n'apporte aucune valeur ajoutée (nouveaux messages) pour la construction du graphe global. Effectivement, après un certain seuil, les services abstraits disponibles commencent à se ressembler vu la limitation des messages qu'ils utilisent. Cette évaluation nous montre que pour agrandir la taille du graphe global, il faut augmenter non seulement le nombre de services abstraits, mais aussi le nombre de messages d'entrée/sortie.

7.4.3. Impact de la variation des événements sur le temps de réponse

Dans ce test, nous avons fixé le nombre de services abstraits à 10^3 et nous avons varié le nombre d'événements (i.e. tâches) qui se produisent dans l'environnement (**Figure 7.30**). Dans la première évaluation, le nombre d'événements survenus varie de 1 à 10 par un pas de 1 événement généré aléatoirement (**Figure 7.30.a**). Les résultats obtenus sont comparés aux trois approches précédentes. Dans la seconde évaluation, nous avons varié le nombre d'événements survenus de 1 à 10^3 par un pas de 10^2 événements générés aléatoirement, et ce, afin de tester la scalabilité de l'algorithme d'extraction du graphe *SubGoSC* (**Figure 7.30.b**).

La **figure 7.30.a** montre que le temps de réponse total pris par l'algorithme de construction du graphe *AGoSC* et l'algorithme d'extraction du graphe *SubGoSC* est également meilleure que celui des trois approches suivantes : *CDSC, FCoSC et HTN-DL*. Dans les **figures** 7.**30.a et 7.30.b**, le temps requis pour la construction du graphe *AGoSC* reste constant même lorsque le nombre de tâches augmente. Cela confirme l'indépendance de cette construction du nombre d'événements qui survient. C'est pour cette raison que la construction du graphe global peut être réalisée dans la phase offline. Ainsi, le temps effectif dont on tient compte pour répondre à un événement c'est le temps pris par l'algorithme d'extraction du graphe *SubGoSC*. La **figure 7.30.b** montre une bonne scalabilité de cet algorithme lorsque le nombre d'évènements augmente. Cela procure alors une meilleure réactivité pour le Framework *FrEvASeC* face aux événements qui surviennent aléatoirement dans l'environnement ambiant. La **figure 7.30.b** montre aussi que le temps d'extraction de sous-graphes pour chaque événement survenu reste raisonnable et augmente d'une façon quasi linéaire. Il est inférieur à 1 seconde pour 100 événements survenus. Nous notons que, dans un environnement réel, il est peu probable que 100 événements se produisent en même temps. Ainsi, la réactivité du Framework *FrEvASeC* face à la détection d'événements est largement acceptable dans des environnements réels.

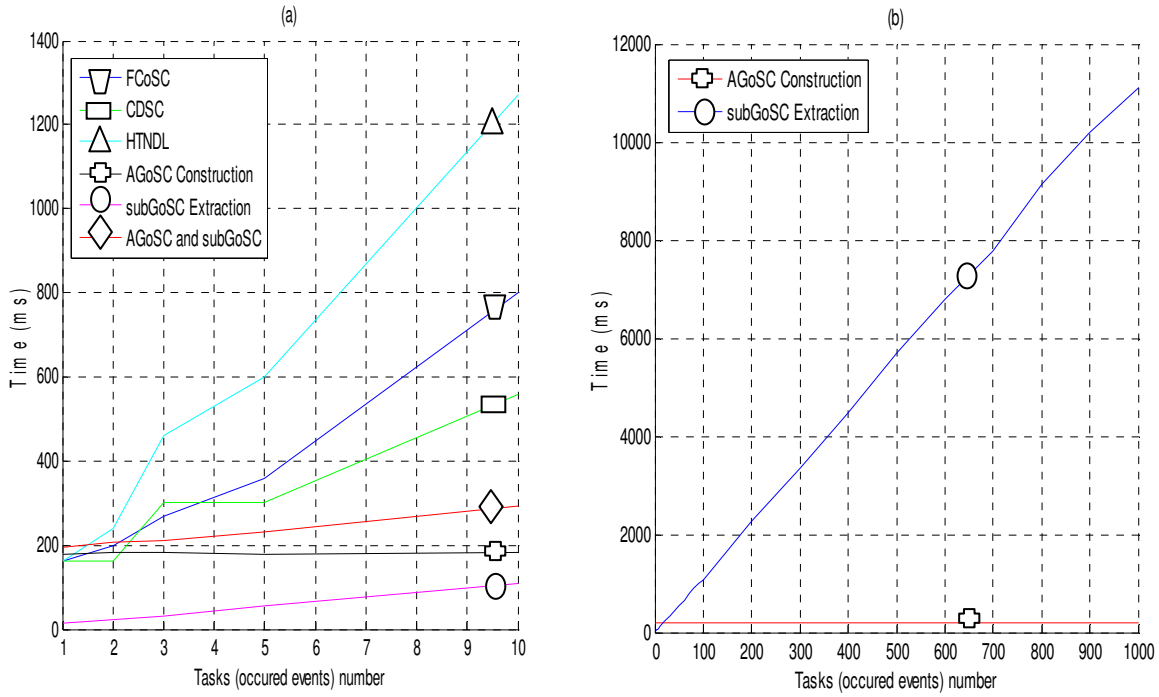


Figure 7.30 : Temps de composition de services vs. Nombre de tâche (i.e. événements): (a) nombre petit de tâches, (b) grand nombre de tâche

7.4.4. Impact de la variation des services concrets sur le monitoring

Dans ce test, le nombre de services abstraits est fixé à 10^3, le nombre d'événements est fixé à 10^2, le seuil d'apprentissage (*learning threshold : LT*) est fixé à 0.1 et le seuil de re-planification (*Replanning Threshold : RT*) est fixé à 5. Nous avons varié le nombre de services concrets générés aléatoirement dans chaque service abstrait de 1 à 40 par un pas de 10 (**Figure 7.31**).

Dans la première évaluation (**Figure 7.31.a**), nous avons mesuré la variation du temps de réponse de l'algorithme de monitoring de services (*Services Monitoring Algorithm : SeMA*). Dans la deuxième évaluation (**Figure 7.31.b**), nous avons mesuré la qualité du monitoring en calculant le taux de succès (*Success Rate*) défini comme étant le nombre d'événements correctement gérés sur le nombre total d'événements survenus. La gestion d'un événement survenu est considérée correcte si on arrive à extraire un sous graphe pour cette événement et à exécuter complètement ce sous graphe. Dans la troisième évaluation (**Figure 7.31.c**), nous avons mesuré le nombre moyen de défaillances survenues dans le niveau inférieur du monitoring (*low level failures*) pour chaque événement détecté. Ces défaillances concernent les pannes des services concrets qui se produisent lors de l'exécution des algorithmes de sélection et d'invocation de services (*Concrete Service Selection et Concrete Service Invocation).* Dans la quatrième évaluation (**Figure 7.31.d**), nous avons mesuré le nombre moyen de défaillances survenues dans le niveau supérieur du monitoring (*high level failures*) pour chaque événement détecté. Ces défaillances concernent les pannes des services abstraits qui se produisent lorsque l'ensemble des services concrets associés à un service abstrait tombent en panne.

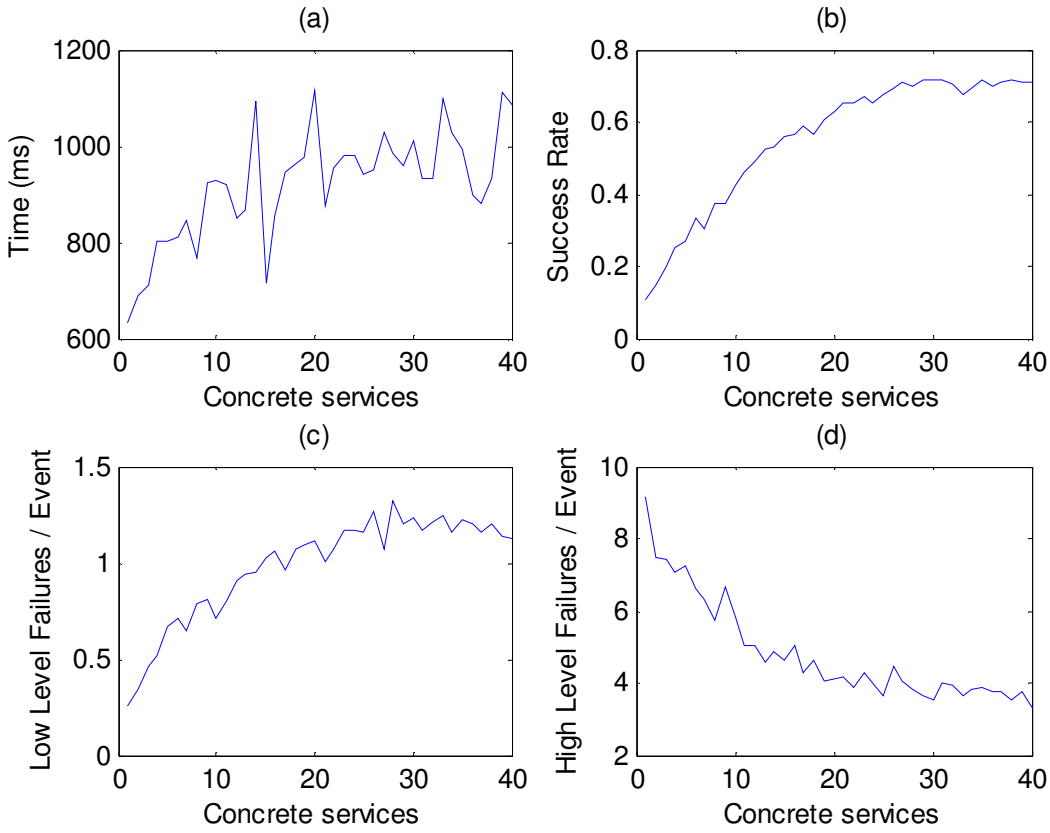


Figure 7.31 : Performance du monitoring de services vs. Nombre de services concrets: (a) temps de monitoring, (b) taux de succès du monitoring, (c) nombre de panes du niveau inférieur du monitoring, et (d) nombre de panes du niveau supérieur du monitoring par événement.

La **figure 7.31.a** montre que, globalement, le temps de monitoring de services requis par l'algorithme *SeMA* augmente lorsque le nombre de services concrets dans chaque service

abstraits augmente. Cela est due au fait que, le temps de sélection d'un service concret augmente avec l'augmentation du nombre de services concrets dans les ensembles de sélection, i.e. les services abstraits. Cependant, cette augmentation n'est pas systématique car nous observons aussi des fluctuations sur le temps de monitoring de services. Cela s'explique par le fait que le temps de monitoring dépend aussi du sous graphe *SubGoSC* extrait pour chaque événement. Ainsi, indépendamment du nombre de services concrets disponibles, lorsqu'un sous graphe contient des services concrets offrant une meilleure qualité de service notamment une grande probabilité de réponse (*PR*), il devient alors fort probable d'obtenir une réponse positive à la première invocation d'un service concret sans avoir besoin de répéter la sélection plusieurs fois. Par conséquent, le temps total de monitoring sera diminué même si le nombre global de services concrets a augmenté, d'où alors l'apparition des fluctuations.

La **figure 7.31.b** montre que le taux de succès (*Success Rate*) augmente lorsque le nombre de services concrets augmente. Ce taux est d'environ 70% lorsque le nombre de services concrets atteint la valeur de 30. Après cette valeur, nous constatons que le taux de succès reste relativement constant même si le nombre de services concrets augmente. Cela signifie que, en augmentant le nombre de services concrets n'implique pas nécessairement l'augmentation du taux de succès. Ceci s'explique par le fait que le processus d'apprentissage défini au niveau de l'algorithme de sélection (*Concrete Service Selection*) s'arrête dés qu'il ait un service concret qui a atteint le seuil d'apprentissage initialement fixé. Ainsi, certains services concrets n'auront aucune chance d'être explorés et invoqués en raison de cette valeur fixée pour le seuil d'apprentissage. Donc, pour mieux explorer un grand nombre de services concrets disponibles, il est nécessaire de réajuster ce seuil apprentissage. Ce réajustement fera l'objet du prochain test effectué à **la section 4.5.**

La **figure 7.31.c** montre que le nombre de défaillances survenues dans le niveau inférieur du monitoring (*low level failures*) par chaque événement détecté augmente d'une façon fluctueuse lorsque le nombre de services concrets augmente. Ceci s'explique par le fait que le niveau inférieur de monitoring tient compte de la réponse obtenue à partir de l'invocation des services concrets. Ainsi, lorsque le nombre de services concrets augmente, il devient très probable que le nombre de réponses négatives augmente aussi et par conséquent, le nombre de défaillances enregistrées au niveau inférieur de monitoring augmente. Mais, comme il reste une faible probabilité que le nombre de réponses négatives diminue légèrement, cela provoque les fluctuations illustrées sur la **figure 7.31.c**. Cette figure montre également qu'en moyenne, il ya un service concret qui tombe en panne pour chaque événement détecté.

La **figure 7.31.d** montre que le nombre de défaillances survenues dans le niveau supérieur du monitoring (*high level failures*) par chaque événement détecté diminue d'une façon fluctueuse lorsque le nombre de services concrets augmente. Ceci s'explique par le fait que, le niveau supérieur de monitoring tient compte de la réponse des services abstraits au lieu de celle des services concrets. Ainsi, lorsque le nombre de services concrets augmente, il y aura plus de chance qu'un service abstrait fourni une réponse positive. Ainsi, le nombre de défaillances du niveau supérieur du monitoring par chaque événement diminue. Mais, comme il ya aussi une faible probabilité qu'un service abstrait fourni une réponse négative malgré l'augmentation du nombre de services concrets, alors les défaillances peuvent augmenter légèrement, cela

provoque les fluctuations illustrées sur la **figure 7.31.d**. Cette figure montre également qu'en moyenne, il ya quatre services abstraits qui tombent en panne pour chaque événement détecté.

7.4.5. Impact de la variation du seuil d'apprentissage sur le monitoring

Dans ce test, nous avons mesuré exactement les mêmes paramètres du test précédent (*temps de monitoring, succès rate, low level failures et high level failures*) mais en faisant varier, cette fois-ci, le seuil d'apprentissage (*learning threshold : LT*). Pour ce faire, nous avons d'abord fixé le nombre de services abstraits à 10^3, le nombre de services concrets dans chaque service abstrait à 30 et le nombre d'événements détectés à 10^2. Ensuite, nous avons fait varier la valeur du seuil d'apprentissage de 10^{-5} à 10^{-1} (**Figure 7.32**).

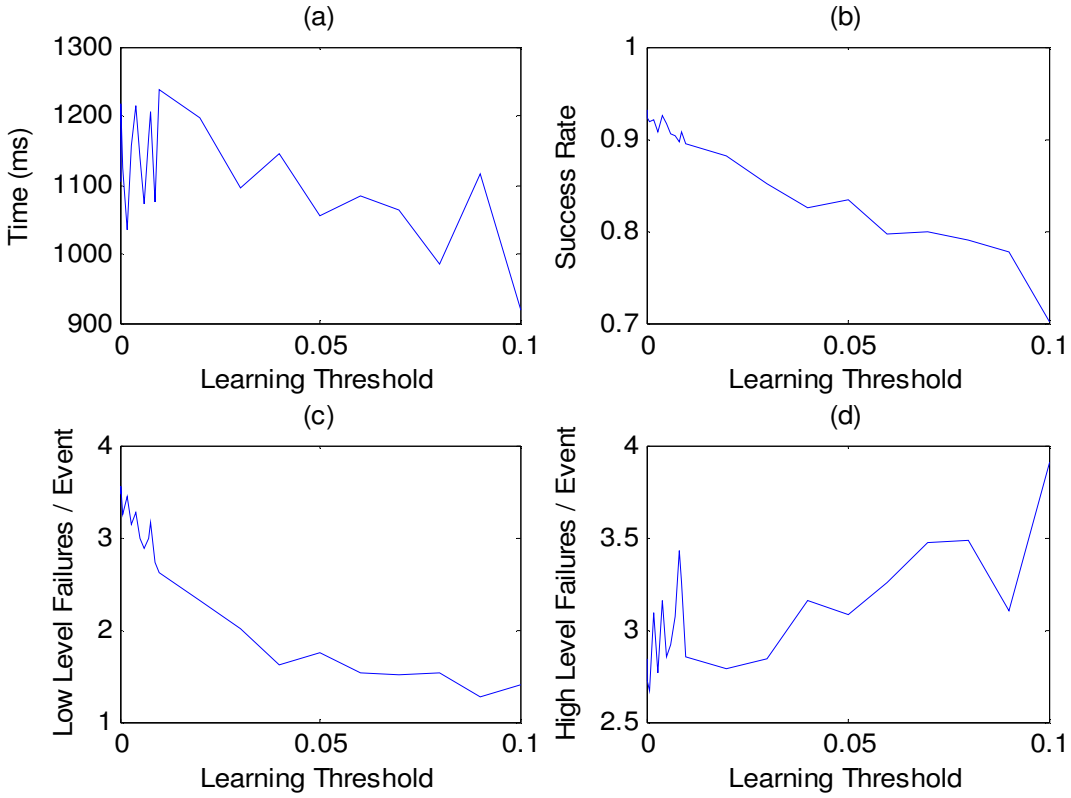


Figure 7.32 : Performance du monitoring de services vs. Seuil d'apprentissage: (a) temps de monitoring, (b) taux de succès du monitoring, (c) nombre de panes du niveau inférieur du monitoring, et (d) nombre de panes du niveau supérieur du monitoring par événement.

La **figure 7.32.a** montre que le temps de monitoring de services requis par l'algorithme *SeMA* diminue à mesure que le seuil d'apprentissage augmente. Cela s'explique par le fait que le processus d'apprentissage appliqué par l'algorithme de sélection de services (*Concrete Service Selection*) prend plus de temps lorsque le seuil d'apprentissage diminue. En effet, lorsque le seuil d'apprentissage diminue, il y a une forte chance qu'un service concret soit invoqué plusieurs fois et d'où la hausse du temps de monitoring.

La **figure 7.32.b** montre que le taux de succès (*Success Rate*) augmente à mesure que le seuil d'apprentissage diminue. Ceci s'explique par le fait que lorsque le seuil d'apprentissage diminue, nous donnons plus de chance à tous les services concrets pour s'exécuter et de ce

fait, nous apprenons davantage l'environnement. Il sera alors fort probable d'obtenir une réponse positive de la part des services concrets invoqués d'où l'augmentation du taux de succès. On constate que ce taux est d'environ 85% lorsque le seuil d'apprentissage égale à 0.05. Pour rappel, pour un même nombre de services concrets (i.e. 30 services), nous avons obtenu un taux de succès égal à 70 dans le test précédent (seuil égal à 0.1). On constate bien l'influence simultanée du nombre de services concrets et du seuil d'apprentissage sur le taux de succès. Par conséquent, pour avoir un taux de succès élevé, il faut d'une part, augmenter le nombre de services concrets disponibles et d'autre part, donner plus de chance à ces services pour qu'ils soient exécutés en diminuant davantage le seuil d'apprentissage. Mais cette diminution aura certainement un impact sur le temps requis pour le monitoring.

La **figure 7.32.c** montre que le nombre de défaillances survenues dans le niveau inférieur du monitoring (*low level failures*) par chaque événement détecté diminue avec quelques fluctuations à mesure que la valeur du seuil apprentissage augmente. Ceci s'explique par le fait que, lorsque le seuil d'apprentissage augmente, le nombre de services concrets invoqués diminue. Donc, moins d'invocation de services implique moins de réponses négatives. Par conséquent, il y a une baisse dans le nombre de défaillances enregistrées.

La **figure 7.32.d** montre que le nombre de défaillances survenues dans le niveau supérieur du monitoring (*high level failures*) par chaque événement détecté augmente avec quelques fluctuations à mesure que la valeur du seuil d'apprentissage augmente. Ceci s'explique par le fait que, lorsque le seuil d'apprentissage augmente, le nombre de services concrets invoqués diminue. Par conséquent, lorsqu'il n'y a pas suffisamment d'invocation de services concrets pour un certain service abstrait, il devient plus probable que ce service abstrait tombe en panne. Ainsi, le nombre de défaillances dans le niveau supérieur du monitoring augmente.

D'une façon générale, on constate que lorsque le paramètre *low level failures* diminue alors le paramètre *high level failures* augmente et vice versa. En effet, la diminution du nombre de pannes du niveau inférieur de monitoring (*low level failures)* signifie soit le nombre de services concrets est petit ou bien le seuil d'apprentissage est grand. Ces deux cas de figure impliquent une augmentation du nombre de pannes du niveau supérieur de monitoring (*high level failures)* car il ya une réelle insuffisance en terme d'invocation de services concrets.

Par ailleurs, nous avons constaté que le taux de succès est d'environ 85% lorsque le seuil d'apprentissage égal à 0.05 et le nombre de service concrets égal à 30. Ce résultat montre clairement la qualité de monitoring des services lors de la phase d'exécution des sous graphes. Pour mieux voir l'intérêt du monitoring des services, nous avons comparé le taux de succès entre plusieurs variantes de l'algorithme du monitoring de services (*SeMA*). Nous savons que cet algorithme combine la sélection et le remplacement de services en cas de pannes. Nous avons alors implémenté deux variantes de cet algorithme pour mesurer l'apport de la sélection et du remplacement sur le taux de succès. Une première variante de l'algorithme *SeMA* n'intègre ni la sélection, ni le remplacement en cas de pannes. Cette variante se base sur un choix aléatoire d'un service concret appartenant à un service abstrait. Le service concret choisi ne sera pas remplacé en cas de panne. La deuxième variante de l'algorithme *SeMA* intègre uniquement l'algorithme de sélection de services (*Concrete Service Selection*) mais sans faire de remplacement en cas de panne des services abstraits. Nous avons alors comparé

le taux de succès entre l'algorithme *SeMA* et ses deux autres variantes implémentées. Cette comparaison est effectuée en fixant le seuil d'apprentissage à 0.05 et le nombre de services concrets à 30. Les résultats obtenus pour cette évaluation sont représentés sur le graphe de la **figure 7.33**.

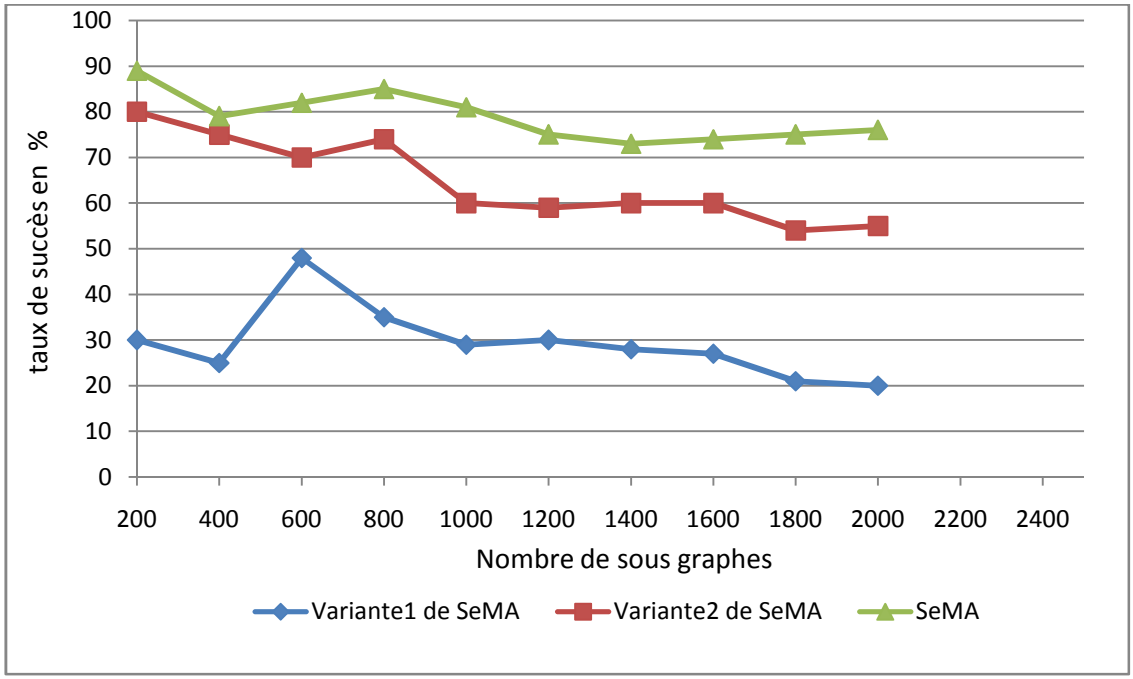


Figure 7.33 : Intérêt du monitoring en termes de taux de succès

Nous constatons que le taux de succès dans la première variante de l'algorithme *SeMA* ne dépasse pas les 50%. Ce taux descend même jusqu'à 20%. Cela signifie qu'il ya environ 80% de sous graphes, i.e. d'événement, qui ne sont pas gérés correctement. Nous remarquons également que le taux de succès dans la deuxième variante de l'algorithme *SeMA* augmente significativement et avoisine en moyenne les 60%. En effet, le processus de sélection favorise les services concrets disponibles qui possèdent une grande probabilité de réponse. Donc, on aura une plus grande chance que le service choisi s'exécute normalement. Même lorsque le service choisi ne s'exécute pas, un remplacement de ce service est effectué automatiquement par le processus de sélection (niveau inférieur de monitoring). Toutefois, si tous les services concrets d'un certain service abstrait ne répondent pas, ce service abstrait ne peut pas être remplacé par le mécanisme de sélection. Donc, le taux de succès fourni par cette deuxième variante reste, comme même, inférieur par rapport à celui fourni par l'algorithme *SeMA*. Nous remarquons que l'utilisation de cet algorithme fait que le taux de succès remonte à une valeur qui avoisine, en moyenne, les 80%. Cela augmente ainsi le nombre de sous graphes exécutés avec sucées. Cette augmentation par rapport à l'algorithme de sélection est due au niveau supérieure de monitoring intégré par l'algorithme *SeMA* qui consiste à remplacer les services abstraits défaillants soit directement par d'autres services abstraits ou bien par une recomposition de services. Etant donné que l'environnement ambiant est fortement instable, l'intégration du monitoring est indispensable pour un système de composition de services afin de maintenir un taux de succès acceptable et gérer convenablement les événements en augmentant les chances d'exécution de leurs sous graphes.

7.5. Synthèse

Dans ce chapitre, nous avons présenté la mise en œuvre des modèles de composition, de sélection et de monitoring des services proposés dans les **chapitres 5 et 6**, avec l'objectif de montrer leurs intérêts et d'évaluer leurs faisabilités. En effet, ces modèles constituent le noyau de notre Framework *FrEvASeC* pour la gestion des événements dans un environnement ambiant. Ainsi, une bonne performance de ces modèles procure une meilleure qualité et réactivité pour le Framework *FrEvASeC*.

Tout d'abord, concernant le modèle de connaissances proposé dans le **chapitre 5**, nous avons montré qu'il s'agit d'un modèle qui tient compte de toutes les entités présentent dans un environnement ambiant : dispositifs, services, messages, etc. Ce modèle s'adapte ainsi parfaitement à la modélisation des environnements ambiants. A travers notamment la possibilité d'établir des règles événementielles, le modèle de connaissances offre des points de jonctions entre d'une part, les objectifs (intentions) des utilisateurs lors d'apparition d'éventuels événements dans l'environnement et d'autre part, les différentes configurations (composition) possibles des services disponibles. Ces règles événementielles sont à la fois des déclencheurs et des guides pour notre système de composition de services. En effet, l'événement spécifié par une règle événementielles constitue un déclencher pour notre système, tandis que l'objectif spécifié par cette même règle constitue un repère pour guider la composition de services. A travers les différents scénarii étudiés, nous avons montré que, grâce à l'utilisation des règles événementielles par le mécanisme d'extraction des sous graphes guidé par des objectifs précis, il n'est pas nécessaire d'indiquer, à priori et d'une façon statique, au système les services à composer. C'est le système qui se charge d'identifier, d'une façon autonome, les services appropriés pour atteindre les objectifs spécifiés. Ainsi, les différents scénarii illustrés sont générés d'une façon automatique et dynamique.

Par ailleurs, nous avons montré que, grâce au mécanisme de construction d'un graphe global, tous les services composables sont initialement identifiés et sauvegardés sous forme d'un graphe. Ce dernier est construit grâce à l'ontologie globale *AmbiOnt* qui décrit d'une façon claire et compréhensible tous les paramètres encapsulés par les messages d'entrée/sortie des différents services. Puisque le graphe global est construit dans une phase offline, alors il permet de réduire considérablement le temps de réponse à un événement qui survient dans l'environnement. Dans la phase online, c'est juste le mécanisme d'extraction qui s'exécute afin de répondre à l'événement survenu, par un sous graphe approprié. Ce sous graphe, qui traduit un scénario possible, est pris en charge depuis sa création jusqu'à son exécution finale. Cette exécution se base essentiellement sur les mécanismes de sélection et de monitoring de services afin d'une part, de garantir une meilleure qualité des services exécutés et d'autre part, de tenir compte de la nature dynamique et incertaine de l'environnement ambiant. Le monitoring permet de remplacer les services défaillant et de faire la recomposition de services dans le cas échant afin de garantir une continuité de service pour atteindre l'objectif.

Enfin, la mise en œuvre et l'évaluation empirique de tous les algorithmes proposés et qui constituent le noyau de notre Framework *FrEvASeC* montrent clairement leur apports en termes de scalabilité et de réactivité face à la nature dynamique et incertain de l'environnement ambiant. En termes de scalabilité, les différents tests d'évaluation des

performances ont montré que le temps de réponse de nos algorithmes reste largement acceptable face à une augmentation significative des services disponibles et des événements qui apparaissent dans l'environnement. Ce temps de réponse reste également meilleur par rapport à plusieurs approches proposées dans le domaine de la composition de services. En termes de réactivité, les différents tests ont montré que notre système maintient un taux de succès avoisinant les 80% devant la forte instabilité de l'environnement ambiant. Ce taux montre que la majorité des événements survenus sont gérés convenablement par notre système.

CHAPITRE 8

Conclusions et perspectives

8.1. Bilan des contributions de cette thèse

Dans cette thèse, nous avons mis en place un système AmI capable d'effectuer une composition dynamique de services dans un environnement ambiant toute en tenant compte des exigences et défis des systèmes intelligents ambiants à savoir : *l'intelligence, la sensibilité au contexte, l'auto-adaptation, l'interopérabilité, la transparence et le passage à l'échelle*. Ce système a été réalisé dans un cadre orienté service suivant dix facettes essentielles à savoir : *modèle des connaissances, architecture générale, modèle de découverte des services, modèle de matching des services, modèle de composition des services, modèle d'évaluation de la QoS et du contexte, modèle de sélection des services*, *modèle de monitoring des services*, *modèle de communication* et *modèle d'évaluation*. Chaque facette permet de répondre à une exigence des systèmes AmI. Dans ce cadre, cette thèse a apporté des contributions significatives pour la réalisation de chaque facette. **La table 8.1** ci-dessous donne une description de chaque facette et un récapitulatif des méthodes et techniques utilisées pour leurs mises en œuvre.

N°	Modèles (facettes du système AmI)	Descriptions
1	*Modèle des connaissances*	- Quatre entités principales : les services, les événements, le contexte et les utilisateurs. - Modélisation des services concrets et des services abstraits. - Séparation entre trois catégories de services concrets : *services sensibles aux événements*, *services sensibles au contexte* et *services de control des dispositifs*. - Deux sous catégories de *services sensibles au contexte* : *services d'acquisition du contexte* et *services de traitement du contexte.* - Modélisation de la qualité de service statique et dynamique. - Association de règles événementielles aux événements. - Modélisation du contexte sous une forme simple par des paramètres de contexte et sous une forme structurée par des messages de contexte. - Les paramètres de contexte représentent des concepts appartenant à une ontologie globale partagée par les services de l'environnement ambiant. - Les messages de contexte sont des structures de données qui encapsulent des paramètres de contexte. Ces messages sont produits et consommés par les différents services concrets. - Les utilisateurs sont pris en compte via leurs spécifications que ce soit en termes des préférences de qualité de service ou bien en termes de règles événementielles.

2	*Architecture générale*	- Modèle multicouche - Structuré en cinq couches principales : *couche physique*, *couche connectivité*, *couche gestion des ressources*, *couche services*, et *couche contrôle*. - Principe général de fonctionnement repose sur une séparation claire entre les rôles des trois catégories de services: *services sensibles aux événements*, *services sensibles au contexte* et *services de control des dispositifs*. - Configurable par les utilisateurs et réactif aux événements. - Fonctionnement en quatre étapes principales : spécifications des utilisateurs, détection des événements, gestion des événements détectés et notification des utilisateurs par les événements détectés et les objectifs réalisés.
3	*Modèle d'évaluation de la QoS et du contexte*	- Evaluation de la qualité de service statique - Evaluation de la qualité de service dynamique - Evaluation de la qualité de service globale
4	*Modèle de matching de services*	- Descriptions des paramètres de contexte dans une ontologie globale. - Similarité sémantique entre les paramètres de contexte. - Similarité sémantique entre les messages fournis et requis par les services concrets. - Critère d'équivalence fonctionnelle entre les services concrets. - Identification et classification des services concrets fonctionnellement équivalents dans des services abstraits.
5	*Modèle de sélection de services*	- Sélection d'un service concret appartenant à un service abstrait selon sa qualité de service globale. - Invocation des services concrets et mise à jour de ses paramètres de qualité de service dynamique. - Sélection de services avec apprentissage Bayésien.
6	*Modèle de découverte de services*	- Architecture à plusieurs répertoires locaux de services. - Utilisation d'un répertoire global des services concrets - Découverte dynamique par mécanismes de souscription /notification. - Mode de découverte hybride : actif et passif.
7	*Modèle de composition de services*	- Représentation graphique des services abstraits - Règles de raisonnement sur les entrées et les sorties des services abstraits. - Représentation d'une composition de services sous formes d'un graphe abstrait. - Planification à base de règles pour la construction et l'extraction des graphes de composition de services. - Construction automatique d'un graphe global de services en offline. - Extraction automatique des sous graphes de services en online.
		- Deux mécanismes de substitution de services : mécanisme de substitution local et mécanisme de substitution global. - Approche multi-niveaux pour le monitoring de services.

8	*Modèle de monitoring de services*	- Niveau initial de monitoring de services. - Niveau supérieur de monitoring de services. - Niveau inférieur de monitoring de services.
9	*Modèle de communication*	- WSDL, SOAP, DPWS
10	*Modèle d'évaluation*	- Une maison intelligente comme environnement ambiant d'expérimentation. - Application dans le domaine d'assistance et de maintien à domicile des personnes âgées à forte dépendance. - Implémentation de plusieurs scénarii prototypiques permettant de contrôler la maison intelligente et d'apporter les services nécessaires aux personnes qui s'y trouvent. - Certains scenarii consistent à surveiller une personne âgée afin d'anticiper et de réduire les risques d'accidents liés au vieillissement et d'effectuer une prévention médicale suite à une évolution de son état de fragilité. - D'autres scenarii portent sur l'assistance en générale : reconnaitre l'activité d'une personne et détecter les mouvements suspects au niveau de la maison intelligente par la vidéo et le son ; maintenir les conditions ambiantes dans cette maison, etc. - La mise en œuvre et l'évaluation empirique des performances de tous les algorithmes proposés.

Table 8.1 : Bilan des contributions de la thèse

La qualité globale de notre système AmI est estimée sur la base des dix critères suivants : *Abstraction de l'hétérogénéité (HA- Heterogeneity Abstraction)*, *Expressivité du modèle de connaissances (KME- Knowledge Model Expressiveness)*, *Evaluation de la QoS et du contexte (QCE- QoS and Context Evaluation)*, *Gestion des incertitudes (UM- Uncertainty Management)*, *Découverte dynamique de services (DSD- Dynamic Service Discovery)*, *Classification dynamique de services (DSC- Dynamic Service Classification)*, *Sélection dynamique de services (DSS- Dynamic Service Selection)*, *Composition automatique et dynamique de services (ADSC- Automatic and Dynamic Service Composition)*, *Auto-adaptation (SA- Self Adapting)*, et *Gestion spontanés des événements (SEH- Spontanious Events Handling)*. La **table 8.2** ci-dessous le dgré de pise en compte de ces dix critères par les différents modèles proposés dans cette thèse.

N°	**Critère de qualité des modèles**	**Prise en compte du critère**
1	*Abstraction de l'hétérogénéité*	✓
2	*Expressivité du modèle de connaissances*	✓
3	*Evaluation de la QoS et du contexte*	✓
4	*Gestion des incertitudes*	✓
5	*Découverte dynamique de services*	✓
6	*Classification dynamique de services*	✓
7	*Sélection dynamique de services*	✓
8	*Composition automatique et dynamique de services*	✓

9	*Auto-adaptation*	✓
10	*Gestion spontanés des événements*	✓

Table 8.2 : La qualité estimée de tous les modèles proposés

8.2. Conclusion générale

Les travaux de recherche présentés dans cette thèse ont fait l'objet de six contributions majeures. Ces différentes contributions pavent des chemins orientés vers la réalisation d'un système AmI capable d'effectuer une composition dynamique de services dans un environnement ambiant. La première contribution a porté sur la proposition d'un modèle des connaissances assez global qui intègre toutes les entités manipulées dans un environnement ambiant. La deuxième contribution a porté sur la proposition d'une architecture générale qui incarne notre vision d'un système AmI. Cette architecture est construite suivant un modèle multicouche afin de supporter un système adaptatif et réactif aux événements. Chaque couche repose sur plusieurs modules interconnectés et engagés dans un processus de collaboration afin de réaliser les objectifs spécifiés lors de la détection des événements dans un environnement ambiant. La troisième contribution a porté sur la proposition d'un modèle de découverte et de classification de services. La découverte et la classification de services se chargent respectivement de localiser et de préparer sémantiquement les services nécessaires à la composition de services. La quatrième contribution a porté sur la proposition d'un modèle de composition de services. Ce modèle permet une construction automatique et dynamique d'un graphe de services en deux phases principales: la phase offline et la phase online. Dans la phase offline, un graphe global reliant tous les services abstraits disponibles est généré automatiquement en se basant sur des règles de décision sur les entrées et sorties des services. Dans la phase online, des sous graphes sont extraits spontanément à partir du graphe global selon les tâches à réaliser qui sont déclenchées par des événements qui surviennent dans l'environnement ambiant. La cinquième contribution a porté sur la proposition d'un modèle de sélection de services. Ce modèle permet l'exécution concrète des graphes de composition de services sur trois phases principales : phase d'estimation de la qualité de service, phase de sélection de services avec apprentissage, et phase d'invocation de services. La sixième et dernière contribution a porté sur la proposition d'un modèle de monitoring de services. Ce modèle permet de contrôler la composition, la sélection et l'invocation de services afin de garantir une continuité de service face aux pannes imprévisibles qui peuvent survenir dans un environnement ambiant. Ces différentes contributions apportées par cette thèse sont testées sur plusieurs scénarii d'assistance et de maintien de personnes à domicile.

Nous avons adopté une démarche incrémentale pour l'élaboration des différentes contributions de cette thèse. Chaque contribution permet de réaliser une facette de notre système AmI appelé *FrEvASeC*. L'ensemble des facettes ainsi réalisées sont structurées sous forme de modules dans une architecture globale configurable par les utilisateurs et réactif aux événements. La démarche suivie et les contributions apportées présentent plusieurs avantages. Premièrement, la structuration modulaire permet de répondre aux besoins de notre système en termes d'évolutivité et de réutilisation. En effet, un module peut être facilement remplacé par un nouveau module sans impacts majeurs sur le fonctionnement du système. Il peut même

être réutilisé pour d'autres besoins applicatifs. Deuxièmement, le modèle des connaissances proposé constitue un support solide pour le système *FrEvASeC* en lui permettant une plus grande ouverture vers les différentes entités de l'environnement ambiant notamment : les services, les événements, le contexte et les utilisateurs. Troisièmement, la découverte dynamique de services permet de palier au problème d'apparition et de disparition des services par une mise à jour dynamique et régulière de la base de tous les services disponibles. Quatrièmement, la classification de services permet de réduire considérablement le temps d'exploration des services concrets à la recherche de certaines fonctionnalités désirées. De plus, l'abstraction des services permet un haut niveau de raisonnement sur les services indépendamment de leurs implémentations concrètes. Cinquièmement, la construction du graphe global des services disponibles est effectuée d'une façon abstraite et en offline, i.e. avant que le système reçoit des tâches à réaliser. Cela permet d'augmenter significativement les performances du système notamment son temps de réponse à une tâche donnée. En outre, le graphe global contient un ensemble de services abstraits qui peuvent avoir plusieurs réalisations concrètes. Ainsi, son exécution peut être adaptée dynamiquement en fonction des services concrets disponibles au moment de l'exécution et en fonction de certains aspects non-fonctionnels relatifs à la qualité de service et au contexte d'utilisation. Par ailleurs, grâce à l'utilisation des règles événementielles par le mécanisme d'extraction des sous graphes guidé par des objectifs précis, il n'est pas nécessaire d'indiquer, à priori et d'une façon statique, au système les services à composer. C'est le système qui se charge d'identifier les services appropriés pour réaliser une tâche donnée. Sixièmement, la sélection et le monitoring de services permettent d'augmenter les chances d'exécution d'un graphe de composition de services dans un environnement dynamique et incertain toute en lui garantissant une meilleure qualité de service.

Enfin, nous avons montré, grâce à plusieurs scénarii d'assistance dans une maison intelligente, l'intérêt et la faisabilité de notre approche dans un environnement ambiant réel. A travers ces scénarii, nous avons montré qu'il est tout à fait possible d'assister l'utilisateur dans plusieurs cas d'utilisation avec les mêmes services et dispositifs. En effet, un même service peut appartenir à plusieurs compositions de services. Chaque composition correspond à une configuration différente des services qui traduit un scénario possible. Grâce à la flexibilité de notre système, les compositions de services sont générées dynamiquement en fonction des tâches à réaliser, et ce, contrairement à des systèmes statiques ou des configurations de services sont initialement spécifiées. Ces systèmes sont rigides et exécutent souvent les mêmes tâches. La panne d'un service peut alors mettre en cause toute une configuration. Par ailleurs, la mise en œuvre et l'évaluation empirique de tous les algorithmes qui implémentent les différentes facettes de notre système *FrEvASeC* montrent clairement leurs apports en termes de scalabilité et de réactivité face à la nature dynamique et incertaine de l'environnement ambiant. En termes de scalabilité, les différents tests d'évaluation des performances ont montré que le temps de réponse de nos algorithmes reste largement acceptable face à une augmentation significative des services disponibles et des événements qui apparaissent dans l'environnement. Ce temps de réponse reste également meilleur par rapport à plusieurs approches proposées dans le domaine de la composition de services. En termes de réactivité, les différents tests ont montré que la majorité des événements détectés

sont gérés convenablement par notre système. Ce dernier maintient un taux de succès acceptable devant la forte instabilité de l'environnement ambiant.

8.3. Les perspectives de ces travaux

Sur la base du travail réalisé, nous pouvons dresser plusieurs perspectives de recherche à court, à moyen et à long terme.

- **Renforcement du degré de collaboration entre les services** : dans l'approche proposée, la communication entre les différents services est établie et gérée par notre système. Comme perspective à court terme, nous envisageons d'interfacer les services par des agents cognitifs capables de raisonner et de communiquer. Dans ce cas, une composition de services se traduit par une collaboration entre plusieurs agents engagés dans un processus de réalisation d'un objectif commun. Ainsi, l'introduction des systèmes multi-agents (SMA) permet d'augmenter les performances de notre système en termes d'exécution des compositions de services. En effet, une fois une composition de services est établie par notre système, les agents concernés seront notifiés afin d'exécuter cette composition d'une manière collaborative.
- **Etude et analyse de la corrélation entre les événements détectés**: notre système traite chaque événement détecté d'une façon individuelle. Toutefois, des corrélations peuvent exister entre les événements qui se produisent dans un environnement ambiant. Par exemple, la détection d'une fumée peut être la cause d'une température très élevée. Dans ce cas, il suffit de traiter l'événement relatif à la détection d'une fumée. Donc, il est nécessaire d'augmenter notre système par des modèles de corrélation entre événements afin d'améliorer l'analyse et l'interprétation des situations et encore réduire le nombre d'événement à traiter.
- **Sécurité et respect de la vie privée** : l'ouverture de l'environnement ambiant à des entités inconnues à priori et à des composants développés par des parties tierces posent des problèmes de sécurité. Il est alors nécessaire de procéder à une authentification systématique des entités de l'environnement ambiant. Il est également nécessaire de gérer les autorisations d'accès de ces entités à des données personnelles des utilisateurs et aussi la sécurisation du protocole d'échange des données entre ces entités.
- **Applications à des scénarii plus complexes** : nous envisageons, comme perspectives à court terme, l'extension de l'ontologie *AmbiOnt* et la mise en œuvre de plusieurs services complémentaires pour valider notre approche dans d'autres cas d'applications. Il s'agit de considérer d'une part, d'autres populations d'usagers comme par exemple les personnes handicapées ou les enfants, et d'autre part, d'autres lieux sociaux tels que les maisons de retraite, les espaces commerciaux, les centres de soins, les espaces urbains, etc.
- **Consolidation de l'aspect sémantique** : notre modèle de connaissances tient compte uniquement des paramètres d'entrée/sorties décrits dans une ontologie globale. Il est nécessaire de tenir compte des pré-conditions et des effets des services par l'extension de l'ontologie globale aux pré-conditions et aux effets des services. Par ailleurs, notre modèle de classification des services se base sur une comparaison entre le contenu des messages. Il est nécessaire d'associer à cette comparaison les relations explicites et implicites qui

existent entre les différents paramètres d'un même message. Cela passe d'abord par la spécification de ces relations dans le modèle de connaissances.

- **Amélioration de la gestion des utilisateurs** : la multiplicité des utilisateurs et de leurs activités simultanées posent aussi le défi de la juste gestion des ressources pour assurer la QoS (qualité de service). En effet, la présence simultanée de plusieurs usagers peut imposer des contraintes supplémentaires en termes de gestion des ressources disponibles. Par exemple, un robot qui est déjà réquisitionné pour assurer un service d'assistance à un usager ne peut pas être sollicité par un autre usager. Dans ce cas, il est nécessaire de définir des priorités d'accès et d'utilisation des ressources disponibles. La présence simultanée de plusieurs usagers peut également dégrader la qualité rendue par certains services notamment ceux d'analyse de son et d'image. En effet, il est plus difficile de reconnaitre une personne au milieu d'une foule que ce soit par sa voix ou ses images. Il est alors nécessaire de faire appel à des services plus performant en terme de traitement de son et d'image afin d'identifier les usagers avec beaucoup plus de précision.
- **Extension des tests d'évaluation des performances** : il s'agit d'effectuer d'autres tests de performances notamment sur les modèles de découverte et de classification de services. Il s'agit aussi d'envisager d'autres tests de comparaisons en introduisant d'autres variantes d'apprentissage.

Références bibliographiques

[1] M. Weiser. “The computer for the 21st century”, *Scientific American*, 1991.

[2] N.A. Streitz and P. Nixon. “The disappearing computer – introduction”, *Communications of the ACM, the Disappearing Computer (special issue)*, 48(3):32–35, march 2005.

[3] G. Privat. “ Des objets communicants à la communication ambiante”, *Les Cahiers du Numérique*, 3(4) :23–44, 2002.

[4] K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten, and J.C Burgelman. “Scenarios of Ambient Intelligence”, *Final Report*, 2001.

[5] IST Advisory Group (ISTAG). "Ambient Intelligence: from Vision to Reality", *European Commission*, 2003.

[6] P. Reignier. “Intelligence Ambiante Pro-Active de la Spécification à l’Implémentation”, *PhD thesis*, Université Joseph Fourier, France, 2010.

[7] IST Advisory Group (ISTAG). “Strategic Orientations & Priorities for IST in FP6”, *European Commission Report*, 2002.

[8] L. Sabri. “Modèles sémantiques, raisonnements réactifs et narratifs, pour la gestion du contexte en intelligence ambiante et en robotique ubiquitaire”, *Thèse de doctorat*, Université Paris-Est, France, 2013.

[9] J. H. Kim, Y.D. Kim, and K.H. Lee. “The Third Generation of Robotics : Ubiquitous Robot”, *In Proc. of the 2nd Int. Conf. on Autonomous Robots and Agents*, Palmerston North, New Zealand, 2004.

[10] A. Sanfeliu, N. Hagita, and A. Saffiotti. “Network robot systems”, *Robotics and Autonomous Systems*, volume 56, pages 793–79, 2008.

[11] M. Shiomi, T. Kanda, H. Ishiguro, and N. Hagita. “Interactive humanoid robots for a science museum”, *IEEE Intelligent Systems* 22 (2) 25–32, 2007.

[12] F. Dressler. “Self-organization in autonomous sensor/actuator networks”, *19th IEEE/ACM/GI/ITG Int. Conf on Architecture of Computing Systems - System Aspects in Organic Computing (ARCS'06),* Frankfurt, Germany, Tutorial, 2006.

[13] J. Coutaz, J.L. Crowley. “Plan « intelligence ambiante » : défis et opportunités”, *Document de réflexion conjoint du CNRS et du Groupe de Concertation Sectoriel (GCS3) du Ministère de l’Enseignement Supérieur et de la Recherche*, France, 2008.

[14] J. Bohn, V. Coroama, M. Langheinrich, F. Mattern, and M. Rohs. “Living in a world of smart everyday objects”, *In Social, Economic, and Ethical Implications*, volume 5, pages 763–786. 2004.

[15] S.H. Park, S.H. Won, J.B. Lee, and S.W. Kim. “Smart home - digitally engineered domestic life”, *In : Personal Ubiquitous Computing,* volume 7, pages 189–196., 2003.

[16] H. Pigot, J. Bauchet, and S. Giroux. “Assistive Devices for People with Cognitive Impairments”, *In The Engineering Handbook of Smart Technology for Aging, Disability, and Independence*, chapter 12, pages 217–236. John Wiley & Sons, Inc., 2008.

[17] C. F. Crispim-Junior, V. Joumier, Y. L. Hsu , M. C. Pai, P. C. Chung, A. Dechamps, P. Robert, and F. Bremond. “Alzheimer’s patient activity assessment using different sensors”, *In Gerontechnology*, pages 266-267, 2012.

[18] J. Legon. “’Smart sofa’ aimed at couch potatoes”, *In : CNN,* pages 217–236., 2003.

[19] J. Liberman. “Things That Think”, *MIT press, Massachusetts Institute of Technology*, pages 217–236., Cambridge, MA, USA, 2006.

[20] D.J. Cook and S.K. Das. “How Smart are our Environments ? An Updated Look at the State of the art”, *In Journal of Pervasive and Mobile Computing*, volume 3, pages 53–73. , 2007.

[21] D. J. Philips and S.K. Das. “Vision of the Future”, *Philips Corporate Design*, Eindhoven, the Netherlands, 2006.

[22] M. Gandy, T. Starner, J. Auxier, and D. Ashbrook. "The Gesture Pendant : A Self-Illuminating, Wearable, Infrared Computer Vision System for Home Automation Control and Medical Monitoring", *In Proceedings of the 4th IEEE International Symposium on Wearable Computers*, pages 87–94, Washington, DC, USA, 2000.

[23] R. Kikin-Gil. "BuddyBeads : techno-jewelry for non-verbal communication within teenager girls groups", *Personal Ubiquitous Comput*., volume 2, pages 106–109, 2006.

[24] S. Leonhardt, S. Mukherjee, R. Aarts, R. Roovers, F. Widdershoven, and M. Ouwerkerk "Personal Healthcare Devices," *In, editors, AmIware Hardware Technology Drivers of Ambient Intelligence*, volume 5 of Philips Research, pages 349–370. Springer Netherlands, 2006.

[25] L. Aogán, D. Dermot, and L. Matt. "Point-of-need diagnosis of cystic fibrosis using a potentiometric ion-selective electrode array", *In Analyst*, volume 125, pages 2264–2267, 2000.

[26] J.A. Tamada, M. Lesho, and M.J. Tierney. "Keeping watch on glucose :new monitors help fight the long-term complications of diabetes", *IEEE Spectr*, 39(4) :52–57, 2002.

[27] J. Wahr and K. Tremper. "Non-invasive oxygen monitoring techniques," *Critical Care Clinics*, 11(1) :199–217, 1995.

[28] A. Zahneisen. "SOPHIA- best practice". *Proceedings of the Second German Congress on Ambient Assisted Living*, 2009.

[29] "From research to market : Smart shirt moves", *Georgia Institute of Technology*, GA, USA., 2004.

[30] C. Gopalsamy, S. Park, R. Rajamanickam, and S. Jayaraman. "The Wearable Motherboard : The first Generation of adaptive and responsive textile structures (arts) for medical applications", *Virtual Reality*, volume 4, pages 152–168, 1999.

[31] G. Demiris and J. Tan. "rejuvenating Home Helath Care and Tele-Homecare", *In : J. Tan (Ed.) : E-Health Care Information Systems : An Introduction for Students and Professionals.* Jossey-Bass, San Francisco, CA, USA., pages 267–290, 2005.

[32] P.S. Pandian, K. Mohanavelu, K.P. Safeer, T.M. Kotresh, D.T. Shakunthala, G. Parvati, and V.C. Padaki. "Smart Vest : Wearable multi-parameter remote physiological monitoring system", *in Medical Engineering & Physics*, Volume 30, Issue 4 , Pages 466-477, May 2008 .

[33] P. S. Pandian, K. P. Safeer, G. Pragati, D. T. Shakunthala, B. S. Sundersheshu, and V. C. Padaki. "Wireless sensor network for wearable physiological monitoring", *in Journal of Networks*, Vol 3, No 5, pages 21-29, May 2008..

[34] N. Halin, M. Junnila, P. Loula, and P. Aarnio. "The LifeShirt system for wireless patient monitoring in the operating room", *Journal of Telemedicine and Telecare*, volume 11, pages 41–43, 2005.

[35] M. Scheermesser. "Akzeptanz des Bewegungsmonitorings bei Chronischen Patienten", *In Proceedings of the Second German Congress on Ambient Assisted Living*. VDE, Berlin, Germany, 2009.

[36] D. Schwabe, M. Heide, A. Neudeck, and U. Mehring. "aktive hüftprotektoren mit Telemonitoringfunktion", *In : Proceedings of the German Congress on Ambient Assisted Living*. VDE, Berlin, Germany, 2008.

[37] H. Jagos, J. Oberzaucher, and W. L. Zagler. "Erste Schritte bei der Entwicklung Instrumentierter Schuhe zur sturzvorbeugung alter menschen", *IKTForum*, 2007.

[38] B.J. Munro, J.R. Steele, T.E. Campbell, and G.G.Wallace. "Wearable eHealth Systems for Personalised Health Management : state of the art and future challenges", *Lymberis, Andreas, IOS Press*, Amsterdam, pages 271–277, 2004.

[39] J. Robert and R. Saffiotti. "Navigation by Stigmergy : A realization on an RFID Floor for minimalistic robots", *In. Proc of the IEEE Int Conf on Robotics and Automation (ICRA)*, Koba, Japan, pages 245–252, 2009.

[40] E. Demmester, A. Huntemann, D. Vanhooydonck, G. Vanacker, A. Deggest, H. Van Brussel, and M. Nuttin. "Bayesian estimation of wheelchair driver intents : Modeling intents as geometric paths tracked by the driver", *IEEE/RSJ Int Conf on Intelligent Robots and Systems (IROS)*, pages 5775–5780, 2006.

[41] P. Di, J. Huang, K. Sekiyama, and T. Fukuda. “Motion control of intelligent cane robot under normal and abnormal walking condition”, *In RO-MAN, IEEE*, pages 497–502, 2011.

[42] S. Mefoued, S. Mohammed, and Y. Amirat. “Knee joint movement assistance through robust control of an actuated orthosis”, *IEEE International Conference on Intelligent and Robotic Systems (IROS)*, San Francisco USA, pages 1749–1754, 2011.

[43] M. Satyanarayanan. “Pervasive computing : Visions and challenges”, *IEEE Personal Communications*, aug.10-17, 2001

[44] T. Chaari, F. Laforest, and A. Flory. “ Adaptation des applications au contexte en utilisant les services WEB”, *Conference UbiMob’05*, Grenoble, France, 3-6 June, 2005.

[45] A. Bottaro. “Home SOA : composition contextuelle de services dans les réseaux d’équipements pervasifs”, *Thèse de doctorat*, Université joseph Fourier, France, Décembre 2008

[46] J.M Geib, C. Gransart, and P. Merle. “Corba : des concepts à la pratique”, *Dunod Informatique*, September 1999.

[47] S. Vinoski. “Corba : Integrating diverse applications within distributed heterogeneous environments”, *IEEE Communications Magazine*, 35(2), February 1997.

[48] A. Woolrath, R. Riggs, and J. Waldo. “A distributed object model for the Java system”, *Computing Systems*, 9(4) : 291-312, 1996.

[49] E. Frank. “DCOM : Microsoft Distributed Component Object Model”, *Hungry Minds, Inc*, 1997.

[50] R. Dale. “Inside COM, Microsoft Component Object Model”*, Microsoft Press*, 1996.

[51] F. Peschanski and J-P Briot. “Architectures de composants répartis”. *Chapitre 9, Composants : concepts, techniques et outils, Vuibert*, 2005.

[52] G. T. Heineman W. T. Council. “Component-Based Software Engineering, Putting the Pieces Together”, *Addison Weysley*, 2001.

[53] Object Management Group (OMG). “CORBA Components. Specification.” *OMG TC Document orbos/99-02-05* , Rapport Technique, 1999.

[54] Object Management Group (OMG). “CORBA Components 3.0 - The Container Programming Model”, *Rapport Technique numéro formal/02-06-72*, juin 2002.

[55] “Enterprise JavaBeans Specification. Version 2.1”, *Rapport Technique, Sun MicroSystems*, 2001.

[56] .NET. *Microsoft Corp*. http://www.microsoft.com/net.

[57] E. Bruneton, T. Coupaye, J. Stefani. “The Fractal Component Model”, *Rapport Technique numéro2.0-3, The ObjectWeb Consortium*. Février 2004

[58] A. Andersen, G. Blair, V. Goebel and R. Karlsen. “Arctic Beans: Configurable and Re-configurable Enterprise Component Architectures”, *In Work in Progress session at ACM/IFIP/USENIX International Middleware Conference (Middleware'01),* Heidelberg, Allemagne, Novembre 2001.

[59] N. Jennings, K. Sycara, and M. Wooldridge. “A Roadmap of Agent Research and Development”, *In Proc. AAMAS’98*, 1998.

[60] M. Wooldridge and N.R. Jennings. “Intelligent Agents : Theory and Practice”, *The Knowledge Engineering Review,* 10(2) :115–152, 1995.

[61] S. Aknine. “Contribution à l’étude des méthodes de coordination dans les systèmes multi-agents”, *habilitation à diriger des recherches*, LIP6, soutenance 17/11/2006.

[62] T.W. Malone and K. Crowston. “The interdisciplinary study of coordination”. *ACM Comput. Surv.,* 26(1) :87–119, 1994.

[63] T. Finin, R. Fritzson, and D. McKay. “An overview of KQML : A Knowledge Query and Manipulation Language”, *Technical report*, University of Maryland Baltimore Country, 1992.

[64] FIPA Interaction Protocol Library Specification. http://www.fipa.org/ specs/fipa00025/XC00025E.pdf, 2001.

[65] *http://www.service-architecture.com*

[66] M. MacKenzie, K. Laskey, F. McCabe, P. Brown, and R. Metz. "Reference Model for Service-Oriented Architecture 1.0", *Technical Report wd-soa-rmcdl, OASIS. August 2006.*

[67] M. P. Papazoglou. "Service Oriented Computing: Concepts, Characteristics and Directions", *Proceedings of the Fourth International Conference on Web Information Systems Engineering*, 2003.

[68] J. Rao, X. Su. "A Survey of Automated Web Service Composition Methods", *Semantic web Services and web Process Composition (SWSWPC), First International Workshop*, San Diego, CA, USA, pp. 43-54. July 6, 2004.

[69] B.A. Malloy, N.A. Kraft, J.O. Hallstrom and J.M. Voas. "Improving the Predictable Assembly of Service-Oriented Architectures", *IEEE Software, Volume 23, IEEE Computer Society Press*, 2006.

[70] T. Gardner. "An Introduction to Web Services", disponible sur le site: *http://www.ariadne.ac.uk/issue29/gardner* ", 02-October-2001.

[71] D. Austin. "*Web Services Architecture Requirement*", *W3C Working Draft*, disponible sur le site: *http://www.w3.org/TR/wsa-reqs*. 14 Novembre 2002.

[72] T. Berners-Lee, J. Hendler, and O. Lassila. "The Semantic Web", *In: Scientific Americain*, 2001.

[73] A. Chibani. "Intergiciel multi agents orienté web sémantique pour le développement d'applications ubiquitaires sensibles au contexte ". *Thèse de doctorat*, Université Paris XII, Décembre 2006.

[74] T.R. Gruber. "A Translation Approach to Portable Ontologies", *Knowledge Acquisition*, vol.5, n°2, pp.199-220. 1993,

[75] C. Lopez-Velasco. "Sélection et composition de services Web pour la génération d'applications adaptées au contexte d'utilisation ", *Thèse de doctorat*, Université Joseph Fourier, France, Novembre 2008.

[76] D.L. McGuiness, F. van Harmelen. "OWL Web Ontology Language Overview", http://www.w3.org/TR/owl-features/, *W3C Recommendation,* 2004.

[77] I. Horrocks. "DAML+OIL: a description logic for the semantic web", *Bull. of the IEEE Computer Society Technical Committee on Data Engineering*, 25(1):4-9, March 2002.

[78] G. Klyne, J.J. Carroll. "Resource Description Framework (RDF): Concepts and Abstract Syntax", *W3C Recommendation*, 2004.

[79] H. Eero (ed). "Semantic Web Kick-Off in Finland, Vision, Technologies, Research and Applications", *HIT Publications*, 2002.

[80] D. Brickley, R. Guha, et al, "RDF Vocabulary Description Language," http://www.w3c.org/TR/rdf-schema/, *W3C Recommendation*, 2004.

[81] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. "From SHIQ and RDF to OWL: The making of a web ontology language", *J. of Web Semantics*, 1(1):7-26, 2003.

[82] S. McIlraith, D. Martin. "Bringing Semantics to Web Services", *IEEE Intelligent Systems*, vol.18, n°1, pp.90-93, 2003.

[83] D. Fensel, C. Bussler, A. Maedche. "Semantic Web Enabled Web Services", *In: Procs of the 1st International Semantic Web Conference (ISWC 2002)*, LNCS Springer 2002, pp.1-2. Sardinia, Italy, June 2002.

[84] C. Preist. "Goals and Vision, Combining Web Services with Semantic Web Technology", *In: Studer, R., Grimm, S., Abecker A., Coord. Semantic Web Services – Concepts, Technologies, and Applications. Springer*, pp.159-178. Berlin, Heidelberg, 2007.

[85] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, K. Sycara. "OWL-S: Semantic Markup for Web Services", http://www.w3.org/Submission/OWL-S. *W3C Member Submission*, 2004.

[86] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P.F. Patel-Schneider, editors. "The Description Logic Handbook: Theory, Implementation and Applications", *Cambridge University Press*, Cambridge, UK, 555p, 2003.

[87] J. Farrell and H. Lausen. "Semantic Annotations for WSDL and XML Schema", http://www.w3.org/TR/sawsdl/, *W3C Recommendation*, 2007.

[88] R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M-T. Schmidt, A. Sheth, and K. Verma. "Web Service Semantics – WSDL-S", http://www.w3.org/Submission/WSDL-S, *W3C Member Submission*, 2005.

[89] H.S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. "XML Schema Part 1: Structures Second Edition", http://www.w3.org/TR/xmlschema-1/, *W3C Recommendation*, 2004.

[90] H. Fujisawa, K. Aoki, M. Yamamoto, and Y. Fujita. "Estimation of multicast packet loss characteristic due to collision and loss recovery using FEC on distributed infrastructure wireless LANs", *IEEE International Conference on Wireless Communications and Networking (WCNC'04)*, Atlanta, GA, USA, March 2004.

[91] V. Hourdin, S. Lavirotte, and J-Y. Tigli. "Comparaison des systems de sevices pour dispositifs", *Rapport de recherche*, laboratoire I3S, France, 2006.

[92] S. Kalasapur, M. Kumar, and B.A. Shirazi. "Dynamic Service Composition in Pervasive Computing", *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 7, pp. 907-918, July 2007.

[93] S. Ben Mokhtar. "Intergiciel Sémantique pour les Services de l'Informatique Diffuse", *Thèse de doctorat*, Université de Paris 6, 2007.

[94] N. Milanovic and M. Malek. "Current Solution for Web Service Composition", *IEEE Internet Computing*, Vol. 8, No.6, pp.51-59, November 2004.

[95] B. Benatallah, R. Dijkman, M. Dumas, and Z. Maamar. "Service Composition: Concepts, Techniques, Tools and Trends", *In: Z. Stojanovic, A. Dahanayake, Eds. Service-Oriented Software Engineering: Challenges and Practices*. Idea Group Inc (IGI), pp.48-66, 2005.

[96] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. "Web Services: Concepts, Architectures and Applications", *Springer*, 354p, 2004.

[97] F. Casati and M-C. Shan. "Event-Based Interaction Management for Composite Eservices in eFlow", *Information Systems Frontiers,* 4(1), pp. 19-31, 2002.

[98] P. Kellert and F. Toumani. "Les services web sémantiques", *Revue I3 : Information Interaction-Intelligence,* 2006.

[99] S. Chollet. "Orchestration de services hétérogènes et sécurisés", *Thèse de doctorat*, Université Joseph Fourier (Grenoble I), France, Décembre 2009.

[100] J. Yang and M. P. Papazoglou. "Service components for managing the life-cycle of service compositions", *Information Systems*, no. 2, 97–125, 2004.

[101] C. Parent and S. Spaccapietra. "Issues and Approaches of Database Integration", *Communications of the ACM*, 41(5) :166–178, 1998.

[102] A.D. Preece, K. Hui, W.A. Gray, P. Marti, T.J.M. Bench-Capon, D.M. Jones, and Z. Cui. "The KRAFT Architecture for Knowledge Fusion and Transformation", *Knowledge Based Systems*, 13(2-3):113–120, 2000.

[103] Y. Charif and N. Sabouret. "An Overview of Semantic Web Services Composition Approaches", *In Proc. International Workshop on Context for Web Services (CWS'05)*, 2005.

[104] H. Lauren, D. Roman, and U. Keller. "Web Services Modeling Ontology- Standard (WSMO-Standard)". *http://wsmo.org/2004/d2/v0.2/*, March 2004.

[105] S. Ambroszkiewicz. "Agent-based Approach to Service Description and Composition", *In Proc. 1st International Workshop on Radical Agent Concepts (WRAC),* pages 135–149, 2003.

[106] J. Cao, M. Li, S.S. Zhang, and Q. Deng. "Composing Web Services based on Agent and Workflow", *In Proc. of the 2nd International Workshop on Grid and Cooperative Computing (GCC)*, volume 3032, pages 948–955, 2004.

[107] I. Müller, R. Kowalczyk, and P. Braun. "Towards Agent-based Coalition Formation for Service Composition", *In Proc. of the International Conference on Intelligent Agent Technology (IAT'06)*, pages 73–80, 2006.

[108] S. Dustdar and W. Schreiner. “A Survey on Web services Composition”, *International Journal of Web and Grid Services*, 1, 2005.

[109] Y. Charif, “Chorégraphie dynamique de services basée sur la coordination d’agents introspectifs”, *Thèse de doctorat*, Université Pierre et Marie Curie Paris VI, Décembre 2007.

[110] T. Osman, D. Thakker, and D. Al-Dabass. “Bridging the Gap between Workflow and Semantic-based Web services Composition”, *In Proc. of the Web Service Composition Workshop WSCOMPS05*, 2005.

[111] C. Marin, “Une approche orientée domaine pour la composition de services”, *Thèse de doctorat*, Université Joseph Fourier (Grenoble I), France, Mai 2008.

[112] A. Barros, M. Dumas, P. Oaks. “Standards for Web Service Choreography and Orchestration: Status and Perspectives”, *In: Procs of the 3rd International Conference the Business Process Management (BPM 2005), 1st International Workshop on Web Service Choreography and Orchestration for Business Process Management,* pp.1-15, Nancy, France, Sept. 2005

[113] C. Peltz. “Web Services Orchestration and Choreography”, *IEEE Computer*, vol.36, n°10, pp.46-52, 2003.

[114] OASIS. “Web Services Business Process Execution Language Version 2.0”, *http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html*. April 2007.

[115] World Wide Web Consortium. “Ws choreography description language specification”, *http ://www.w3.org/TR/ws-cdl-10/,* November 2004.

[116] World Wide Web Consortium. “Web service choreography interface (wsci) 1.0 specification”, *http ://www.w3.org/TR/wsci/*, 2002.

[117] G. Decker, J. Maria Zaha, and M. Dumas. “Execution semantics for service choreographies”, *3rd International Workshop on Web Services and Formal Methods (WS-FM),* 2006.

[118] J. Yang. “Web service componentization”, *Communications of the ACM*, no. 10, 35–40, 2003.

[119] OSGiTM Alliance. “OSGiTM Service Platform Core Specification, Release 4, Version 4.1”, *http://www.osgi.org/download/r4v41/r4.core.pdf*, April 2007.

[120] C. Escoffier, R. Hall, and Ph. Lalanda. “ipojo : an extensible service-oriented component framework”, *in Proceedings of Services Computing Conference*, 474–481, 2007.

[121] C. Escoffier and R. S. Hall. “Dynamically adaptable applications with ipojo service components”, *6th International Symposium on Software Composition (SC2007)*, March 2007.

[122] N. Ferry. “Adaptations dynamiques au contexte en informatique ambiante : propriétés logiques et temporelles ”, *Thèse de doctorat*, Université de Nice - Sophia Antipolis, France, Décembre 2011.

[123] BEA, IBM, Interface21, IONA, Oracle, SAP, Siebel, and Sybase. “Service component architecture building systems using a service oriented architecture”, *http ://www.osoa.org/download/attachments/28/SCA_White_Paper1_-09.pdf ?version=1*, November 2005.

[124] D. Chappell. “Introducing sca”, *http://www.davidchappell.com/articles/Introducing_SCA.pdf*, 2007.

[125] M. Fowler. “Inversion of control containers and the dependency injection pattern”, *http ://martinfowler.com/articles/injection.html*, 2004.

[126] OSOA Group. “Sca assembly model v1.00 specification”, *http ://www.osoa.org/display/Main/Service+Component+Architecture+Specifications*, March 2007.

[127] V. Hourdin, J-Y. Tigli, S. Lavirotte, G. Rey, and M. Riveill. “SLCA, composite services for ubiquitous computing”, *In Proc. of the 5th Int. Conf. on Mobile Technology, Applications and Systems (Mobility),* September 2008.

[128] J-Y. Tigli, S. Lavirotte, G. Rey, V. Hourdin, and M. Riveill. “Lightweight Service Oriented Architecture for Pervasive Computing”, *IJCSI International Journal of Computer Science Issues*, Vol. 4, No. 1, University of Nice – Sophia Antipolis, France, 2009.

[129] M. Broy, A. Deimel, J. Henn, K. Koskimies, F. Plasil, G. Pomberger, W. Pree, M. Stal, and C. Szyperski. "What characterizes a (software) component?". *Software-Concepts & Tools, Springer,* Vol. 19, pp. 49-56, 1998.

[130] M. Clarke, G. S. Blair, G. Coulson, and N. Parlavantzas. "An efficient component model for the construction of adaptive middleware". *In the Proceedings IFIP Middleware 2001*, pp. 160–178. Springer-Verlag. 2001.

[131] R. Englander. "Developing Java Beans", *O'Reilly \& Associates*, CA, USA, 1997.

[132] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. "Automatic Composition of E-Services that Export Their Behavior," *Proc. 1st Int'l Conf. Service-Oriented Computing (ICSOC 03), LNCS 2910, Springer-Verlag*, pp. 43–58, 2003.

[133] T. Bultan, X. Fu, R. Hull, J. Su. "Conversation Specification: A New Approach to Design and Analysis of E-Service Composition," *Proc. Int'l World Wide Web Conf. (WWW 2003), ACM Press*, pp. 403–410, 2003.

[134] D. Berardi, D. Calvanese, G. De Giacomo, and M. Mecella. "Reasoning about Actions for e-Service Composition", *ICAPS Workshop on Planning for Web Services (P4WS 2003)*, 2003.

[135] S. McIlraith and T. Son. "Adapting Golog for Composition of Semantic Web Services", *KR '02*, 2002.

[136] K. Erol, J. Hendler, and D. S. Nau. "Semantics for Hierarchical Task Network Planning", *UMIACS Technical Report*, 1994.

[137] D. Wu, E. Sirin, J. Hendler, D. Nau, and B. Parsia, "Automatic Web Services Composition Using SHOP2", *Workshop on Planning for Web Services*, 2003.

[138] R. Hamadi and B. Benatallah, "A Petri-Net-Based Model for Web Service Composition," *Proc. 14th Australasian Database Conf. Database Technologies, ACM Press*, pp. 191–200, 2003.

[139] H. Kautz and B. Selman. "Unifying SAT-based and graph-based planning". *In Proc. Of the 16th International Joint Conference on Artificial Intelligence (IJCAI),* pages 318{325, Stockholm, Sweden, 1999.

[140] P. Doshi, R. Goodwin and R. Akkiraju, K. Verma. "Dynamic Workflow Composition using Markov Decision Processes", *International Journal of Web Services Research*, 2(1): 1-17, Jan-March 2005.

[141] M. Vallee, F. Ramparany, and L. Vercouter, "A Multi-Agent System for Dynamic Service Composition in Ambient Intelligence Environments", *The 3rd International Conference on Pervasive Computing PERVASIVE, 2005*.

[142] A. Patil, S. Oundhakar, A. Sheth, and K. Verma. "METEOR-S Web Service Annotation Framework", *The Proceedings of the Thirteenth International World Wide Web Conference*, May 2004.

[143] W3C. **"**Semantic Annotations for Web Services Description Language", *http://www.w3.org/2002/ws/sawsdl/* ,2002.

[144] B. Medjahed, A. Bouguettaya, and A. K. Elmagarmid. "Composing Web Services on the Semantic Web", *The VLDB journal*, September 2003.

[145] S. A. Chun, V. Atluri, and N. R. Adam. "Using Semantics for Policy-Based Web Service Composition", *In Journal Distributed and Parallel Databases*, Volume 18, Issue 1, Pages 37-64, July 2005

[146] L. Chen, N.R. Shadbolt, C. Goble, F. Tao, S.J. Cox, C. Puleston, and P.R. Smart. "Towards a Knowledge-based Approach to Semantic Service Composition", *In Proceedings of 2nd International Semantic Web Conference (ISWC2003)*, 2003.

[147] J. Robinson, I. Wakerman, and T. Owen. "Scooby: Middleware for Service Composition in Pervasive Computing", *MPAC '04 Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing*. Pages 161 – 166. 2004.

[148] W. Zahreddine and Q. H. Mahmoud. "A Framework for Automatic and Dynamic Composition of Personalized Web Services", *Advanced Information Networking and Applications*, AINA 2005.

[149] S. B. Mokhtar, N. Georgantas, and V. Issarny. "COCOA : Conversation- Based service composition for pervasive computing environments", *In ACS/IEEE International Conference on Pervasive Services*, pages 29–38, 2006.

[150] K. Erol, J. Hendler, and D.S. Nau. "HTN Planning: Complexity and Expressivity", *In: Procs of the 12th National Conference on Artificial Intelligence (AAAI 1994)*, *AAAI Press*, pp.1123-1128. Seattle, WA, USA. July-Aug. 1994.

[151] D. McDermott. "Estimated-Regression Planning for Interactions with Web Services", *In: Procs of the 6th International Conference on Artificial Intelligence Planning Systems*, *AAAI 2002*, pp.201-211. Toulouse, France, April 2002.

[152] B. Srivastava and J. Koehler. "Web Service Composition – Current Solutions and Open Problems", *In: Procs of the 13th International Conference on Automated Planning and Scheduling (ICAPS 2003), Workshop on Planning for Web Services,* Trento, Italy, June 2003.

[153] M. Carman, L. Serafini, P. Traverso. "Web Service Composition as Planning", *In: Procs of the 13th International Conference on Automated Planning and Scheduling (ICAPS 2003)*, *Workshop on Planning for Web Services*, Trento, Italy, June 2003.

[154] C. A. Petri. "Kommunikation mit Automaten", *PhD thesis*, Institut für instrumentelle Mathematik, Bonn, 1962.

[155] C. Ouyang, E. Verbeek, W.M.P. van der Aalst, S. Breutel, M. Dumas, and A.H.M. ter Hofstede. "Formal semantics and analysis of control flow in ws-bpel", *Sci. Comput. Program.*, 67 :162–198, July 2007.

[156] X. Yi and K.J. Kochut. "A cp-nets-based design and verification framework for web services composition", In *Proceedings of the IEEE International Conference on Web Services*, ICWS '04, pages 756–, Washington, DC, USA, 2004.

[157] V. Ermolayev, N. Keberle, and S. Plaksin. "Towards Agent-Based Rational Service Composition – RACING Approach", *In M. Jeckle and L-J. Zang, editors, Proc. of the International Conference on Web Services Europe*, volume LNCS 2853, pages 167–182, Springer-Verlag, Erfurt, Germany, September 2003.

[158] J. Kopecky, D. Roman, M. Moran, and D. Fensel. "Semantic web services grounding", *AICT-ICIW'06*, page 127. 19-25 Feb. 2006.

[159] A. Westerinen. "Rfc 3198 : Terminology for policy-based management", *http://www.rfc-base.org/txt/rfc-3198.txt.* 2001.

[160] A. Saffiotti, M. Broxvall, B. Seo, and Y. Cho. "The PEIS-ecology project: a progress report", *ICRA-07 Workshop on Network Robot Systems*, pp. 16–22, Rome, Italy, 2007.

[161] M. Broxvall, B.S. Seo, and W.Y. Kwon. "The PEIS Kernel: A Middleware for Ubiquitous Robotics", *IROS Workshop on Ubiquitous Robotic Space Design and Applications*, 2007.

[162] J.Y. Tigli, M. Riveill, G. Rey, S. Lavirotte, V. Hourdin, D. Cheung, and E. Callegari. "WComp Middleware for Ubiquitous Computing: Aspects and Composite Event-based Web Services", *Annals of Telecommunications*, Volume 64, Numbers 3-4, pp. 197–214 , 2009.

[163] Y.G. Ha, J.C. Sohn, and Y.J. Cho. "Service-oriented integration of networked robots with ubiquitous sensors and devices using the semantic Web services technology", *IEEE/RSJ International Conference on Intelligence Robots and Systems (IROS)*, pp. 3947 – 3952, 2005.

[164] E. Sirin, B. Parsia, and J. Hendler. "Template-based composition of semantic Web services", *AAAI-05 Fall Symposium on Agents and the Semantic Web*, Arlington-Virginia (USA) , 2005.

[165] A. Yachir, K. Tari, Y. Amirat, A. Chibani, and N. Badache. "QoS based framework for ubiquitous robotic services composition", *IEEE/RSJ International Conference on Intelligence Robots and Systems (IROS)*, pp. 2019–2026, 2009.

[166] A. Yachir, K. Tari, A. Chibani, and Y. Amirat. "Towards an automatic approach for ubiquitous robotic services composition", *IEEE/RSJ International Conference on Intelligence Robots and Systems (IROS)*, pp. 3717–3724, 2008.

[167] K. Tari, Y. Amirat, A. Chibani, A. Yachir, and A. Mellouk. "Context-aware dynamic service composition in ubiquitous environment", *IEEE International Communications Conference (ICC'10),* Cape Town, South Africa, pp. 1-6, 2010.

[168] K. Tari, Y. Amirat, A. Chibani, and A. Yachir. "Rule-based approach for automatic service composition in ubiquitous environment", *The 6th International Conference On Ubiquitous Robots And Ambient Intelligence (URAI)*, Gwangju, Korea, October 2009.

[169] N. Ibrahim, F. Le Mouël, and S. Frénot. "MySIM: a Spontaneous Service Integration Middleware for Pervasive Environments", *ACM International Conference on Pervasive Services (ICPS '09)*, London, UK pp. 1-10, 2009.

[170] W. Yu, J-Y. Lee, Y-G. Ha, M. Jang, J-C. Sohn, Y-M. Kwon, and H-S. Ahn. "Design and Implementation of a Ubiquitous Robotic Space", *IEEE Transactions on Automation Science And Engineering*, Vol. 6, No. 4, October 2009.

[171] R. Rouvoy, P. Barone, Y. Ding, F. Eliassen, S. Hallsteinsen, J. Lorenzo, A. Mamelli, and U. Scholz. "MUSIC: Middleware Support for Self-Adaptation in Ubiquitous and Service-Oriented Environments", *Self-Adaptive Systems, LNCS 5525*, pp. 164–182, Springer-Verlag Berlin Heidelberg, 2009.

[172] F. Rosenberg, P. Leitner, A. Michlmayr, P. Celikovic, and S. Dustdar. "Towards Composition as a Service - A Quality of Service Driven Approach", *In: Proceedings of the First IEEE Workshop on Information and Software as Service (WISS'09), co-located with the 25th International Conference on Data Engineering (ICDE'09)*, Shanghai, China, March 2009.

[173] K. Fujii and T. Suda. "Semantics-based dynamic service composition", *IEEE Journal on Selected Areas in Communications*, Vol. 23, No. 12, 2005.

[174] O. Davidyuk, N. Georgantas, V. Issarny, and J. Riekki. "MEDUSA: Middleware for End-User Composition of Ubiquitous Applications*". In: Mastrogiovanni, F. & Chong, N.Y. (Eds.), Handbook of Research on Ambient Intelligence and Smart Environments: Trends and Perspectives. IGI Global (Ed.)*, 2009.

[175] D. Benslimane, A. Arara, G. Falquet, Z. Maamar, P. Thiran, and F. Gargouri. **"**Contextual Ontologies: Motivations, Challenges, and Solutions", In *Fourth Biennial International Conference on Advances in Information Systems*(Ed, Springer) Izmir, Turkey, pp. 168 – 176, 2006.

[176] Q. Z Sheng, B. Benatallah, Z. Maamar, and A. H. H. Ngu. "Configurable Composition and Adaptive Provisioning of Web Services", *IEEE Transactions on Services Computing (TSC),* 2(1), pp. 34-49, 2009.

[177] Z. Maamar, D. Benslimane, P. Thiran, C. Ghedira, S. Dustdar, and S. Sattanathan. "Towards a context-based multi-type policy approach for Web services composition", *Data & Knowledge Engineering, Elsevier Science Publisher,* 62(2), 2007.

[178] K. Boukadi. "Coopération interentreprises à la demande : Une approche flexible à base de services adaptables", *Thèse de doctorat*, École Nationale Supérieure des Mines de Saint Etienne, France, Novembre 2009.

[179] H. Chen, T. Finin, and A. Joshi. "An Ontology for Context-Aware Pervasive Computing Environments", *Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review,* 18(3), pp. 197-207, 2004.

[180] G. Chen and D.A. Kotz. "Survey of Context-Aware Mobile Computing Research", In *Technical Report TR2000-381, Dartmouth*, pp. 16, 2000.

[181] D. Saha and A. Mukherjee. "A Pervasive Computing: a Paradigm for the 21st Century", *Computer,* 36(3), pp. 25-31, 2003.

[182] J-Y. Tigli and S. Lavirotte. "Adaptation dynamique à l'environnement d'exécution : un enjeu pur l'informatique mobile et ambiante", In *Séminaire, Grenoble, 2006.*

[183] B.N. Schilit and M.M. Theimer. "Disseminating active map information to mobile hosts", *Network, IEEE,* 8 (5), pp. 22 – 32, 1994.

[184] S. Jang and W. Who. "Ubi-UCAM: A Unified Context-Aware Application Model", In *the 4th International and Interdisciplinary Conference on Modeling and Using Context* Springer, Stanford, USA, pp. 178-189, 2003.

[185] T. Winograd. "Architectures for Context", *Human Computer Interaction Journal,* 6(2&3), pp. 401-419, 2001.

[186] L. Gong. "Contextual modeling and applications", In *IEEE International Conference on Systems, Man and Cybernetics*, Vol. 1, pp. 381- 386, 2005.

[187] P.J. Brown. "The Stick-e Document : a framework for creating Context-aware applications". *Electronic Publishing '96*, 259-272, 1996.

[188] N. **S.** Ryan, J. Pascoe, and D. R. Morse. "*Enhanced Reality Fieldwork : the Context-Aware Archeological Assistant"*, *In V. Gaffney, M. van Leusen, and S. Exxon, editors, Computer Applications in Archaeology* 1997.

[189] K.A Dey and D.G. Abowd. "Towards a better understanding of context and contextawarness"*, In Computer Human Interactions (CHI2000) Workshop on the What, Who, Where and How of Context Awarness*. 2000.

[190] G. Rey and J. Coutaz. " Le Contexteur : une abstraction logicielle pour la réalisation de systèmes interactifs sensibles au contexte", *In Proceedings of Interaction Homme Machine (IHM2002)*, p. 105112, 2003.

[191] P. Brézillon, C. Kintzig, G. Poulain, G. Privat, and P.N. Favennec. " Expliciter le contexte dans les objets communicants", *Hermès, chapitre 21*, p. 295-303, 2002.

[192] J. Euzenat, J. Pierson, and F. Ramparany. "Dynamic context management for pervasive computing", *INRIA Rhône-Alpes, France Telecom R&D*, 2004 .

[193] B. Schilit, M. Theimer, and B. Welch. "Customising mobile applications", *In Proceedings of USENIX Symposium on Mobile and Location-Independent Computing*, pages 129–138, August 1993.

[194] A. Schmidt, K.A. Aidoo, A. Takaluoma, U. Tuomela, K.V. Laerhoven, and W.V. d. Velde. "Advanced Interaction in Context", In *the 1st international symposium on Handheld and Ubiquitous Computing*(Ed, Springer) London, UK, pp. 89-101, 1999.

[195] K. Henricksen, J. Indulska, and A. Rakotonirainy. "Modeling context information in pervasive computing systems", *In Pervasive 2002*, pages 167–180, Zurich, Switzerland, 2002.

[196] Q.Z. Sheng and B. Benatallah. "ContextUML : A UML-Based Modeling Language for Model-Driven Development of Context-Aware Web Services", In *the 4th International Conference on Mobile Business (ICMB'05), IEEE Computer Society*Sydney, Australia, pp. 11-13, 2005.

[197] H. Chen, F. Perich, D. Chakraborty, T. Finin, and A. Joshi. "Intelligent Agents Meet Semantic Web in a Smart Meeting Room", In *the Third International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS 2004),* New York City, NY, 2004.

[198] D. Preuveneers and Y. Berbers. "Semantic and syntactic modeling of componentbased services for context-aware pervasive systems using owl-s", *In First International Workshop on Managing Context Information in Mobile and Pervasive Environments*, pages 30–39, 2005.

[199] X.H. Wang, D.Q. Zhang, T. Gu, and H.K. Pung. "Ontology Based Context Modeling and Reasoning using OWL", *In PerCom Workshops*, pp. 18-22, 2005.

[200] G. K. Mostéfaoui, G. Pasquier-Rocha, and P. Brézillon. "Context-aware computing: A guide for the pervasive computing community ". *In ICPS*, pages 39–48, 2004.

[201] P. J. Brown. "Triggering information by context". *In personnal Technologies*. v. 2, p. 19. 1998.

[202] A. Chibani, K. Djouani, Y. Amirat. "Agents-middleware approach for context awareness in pervasive computing", *ICEIS 2003, Proceedings of the 5th International Conference On Enterprise Information Systems*, Vol. 4, Angers, France, pp.184-189, 23-26 April 2003.

[203] G.D. Abowd, A.K. Dey, P.J. Brown, N. Davies, M. Smith, and P. Steggles. "Towards a Better Understanding of Context and Context-Awareness", In *the 1st international symposium on Handheld and Ubiquitous Computing*Springer-Verlag Karlsruche, Germany pp. 304 – 307, 1999.

[204] P-C. David. "Développement de composants Fractal adaptatifs: un langage dédié à l'aspect d'adaptation", Thèse de doctorat, École des mines de Nantes, France, 2005.

[205] N.B. Behlouli, C. Taconet, and G. Bernard. "An architecture for supporting Development and Execution of Context-Aware Component applications", In *IEEE International Conference on Pervasive Services* Lyon, France, pp. 26-29, 2006.

[206] N. Yahiaoui, B. Traverson, and N. Levy. "*Classification and comparison of adaptable platforms"*. In First International Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT'04) held in conjunction with ECOOP '04, Oslo, Norway, June, 2004.

[207] M. Aksit and Z. Choukair. *"Dynamic, adaptive and reconfigurable systems overview and prospective vision"*. In *ICDCS Workshops 2003*, pages 84–94, 2003.

[208] B. Smith. "Procedural reflection in programming languages", PhD thesis, Massachussetts Institute of Technimogy, USA, 1982.

[209] G. Kiczales. *"Aspect-oriented programming. Surveys",* 28A(4), December 1996.

[210] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. Griswold. "Overview of AspectJ", In *the European Conference on Object Oriented Programming (ECOOP 01)*Budapest, Hungary, 2001.

[211] *Jena2. http ://jena.sourceforge.net/.*

[212] M. Vallee, F. Ramparany, and L. Vercouter. "Flexible Composition of Smart Device Services", In *The 2005 International Conference on Pervasive Systems and Computing (PSC-05),* Las Vegas, Nevada, USA., June (2005), pp. 27-30.

[213] C. Hesselman, A. Tokmako, P. Pawar, and S. Iacob. "Discovery and composition of services for context-aware systems", volume 4272, 2006.

[214] H. Pourreza, P. Graham, On the fly service composition for local interaction environments, In IEEE International Conference on Pervasive Computing and Communications Workshops, page 393. IEEE Computer Society, 2006.

[215] W. L. C. Lee, S. Ko, S. Lee, and A. Helal. "Context-aware service composition for mobile network environments", *In 4th International Conference on Ubiquitous Intelligence and Computing* (UIC2007), 2007.

[216] A. Mingkhwan, P. Fergus, O. Abuelma'Atti, M. Merabti, B. Askwith, M. B. Hanneghan. "Dynamic service composition in home appliance networks", *Multimedia Tools and Applications*, 29(3):257-284, 2006.

[217] U. Saif, H. Pham, J.M Paluska, J. Waterman, C. Terman, and S. Ward. "A case for goal-oriented programming semantics", *In Workshop on System Support for Ubiquitous Computing (UbiSys'03),* 5th International Conference on Ubiquitous Computing (UbiComp 2003), Seattle, WA, USA, October 12, 2003.

[218] S. B. Mokhtar, N. Georgantas, and V. Issarny. "Ad hoc composition of user tasks in pervasive computing environments", *In Proceedings of the 4th workshop on software composition*, LNCS3628. 2005.

[219] Z. Maamar, S. K. Mostefaoui, and H. Yahyaoui. "Toward an agent-based and context-oriented approach for web services composition". *IEEE Transactions on Knowledge and Data Engineering*, 17(5):686–697, 2005.

[220] A.B. Hassine, S. Matsubara, and T. Ishida. "A Constraint-based Approach to Horizontal Web Service Composition", *Language Grid Project, National Institute of Information and Communications Technology*, NTT Communication Science Laboratories, and Kyoto University. 2006.

[221] L. Qiu, Z. Shi, and F. Lin. "Context Optimization of AI planning for Services Composition", In *the IEEE International Conference on e-Business Engineering (ICEBE 2006), October 2006, Shangai, China*, 2006.

[222] L. Qiu, Z. Shi, F. Lin, and Z. Shi. "Context Optimization of AI Planning for Semantic Web Services Composition", *Service Oriented Computing and Applications,* 1(2), pp.117-128. 2007.

[223] K. Nahrstedt, D. Xu, D. Wichadakul, and B.Li. "Qos-aware middleware for ubiquitous and heterogeneous environments", *IEEE Communications magazine*, 39(11):2–10, 2001.

[224] M. Alrifai, T. Risse, P. Dolog, and W. Nejdl. "A scalable approach for QoS-Based web service selection", *in Proc. Workshop Service-Oriented Computing (ICSOC'08)*, pp. 190–199, 2008.

[225] N. Ben Mabrouk, S. Beauche, E. Kuznetsova, N. Georgantas, and V. Issarny. "QoS-Aware service composition in dynamic service oriented environments", *Middleware 2009*, pp. 123–142, 2009.

[226] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar. "End-to-End Support for QoS-Aware Service Selection, Binding, and Mediation in VRESCo", *IEEE Transactions on Services Computing*, vol. 3, no. 3, July - September 2010.

[227] P. Leitner, A. Michlmayr, F. Rosenberg, and S. Dustdar. "Selecting Web Services Based on Past User Experiences", *IEEE Asia-Pacific Services Computing Conference (IEEE APSCC)*. 2009

[228] A. Mello Ferreira, K. Kritikos, B. Pernici. "Energy-Aware Design of Service-Based Applications", *Lecture Notes in Computer Science*, Volume 5900/2009, 99-114, DOI: 10.1007/978-3-642-10383-4_7 , Springer, 2009.

[229] A. Charfi and M. Mezini, "AO4BPEL: An Aspect-oriented Extension to BPEL", *World Wide Web,* **10**(3), pp. 309-344. 2007.

[230] A. Charfi, R. Berbner, M. Mezini, and R. Steinmetz. "Management Requirements of Web Service Compositions", In *the 2nd ECOWS07 Workshop on Emerging Web Services Technology, WEWST,* Germany, *2007*.

[231] A. Charfi. "Aspect-Oriented Workflow Management", *Concepts, Languages, Applications, (Ed, Mueller, V. V. D.),* 2008.

[232] A. Charfi, T. Dinkelaker, and M. Mezini. "A plug-in architecture for self-adaptive web service compositions". *In IEEE International Conference* on *Web Services. ICWS 2009.*, pages 35–42, 2009.

[233] M. B. Hmida, R.F. Tomaz, and V. Monfort. "Applying AOP Concepts to Increase Web Services Flexibility", In *the proceedings of the IEEE International Conference on Next Generation Web Services Practices (NWeSP 05)*Seoul, Korea, pp. 169-174, 2005.

[234] M. B. Hmida, C. Boutrous-Saab, S. Haddad, V. Monfort, and R.F. Tomaz. "Towards the Dynamic Adaptability of SOA", In *the Ninth International Conference on Enterprise Information Systems (ICEIS)*Funchal, Madeira, Portugal, pp. 474-479, 2007.

[235] R. F. Tomaz, M. B. Hmida, and V. Monfort. "Concrete Solutions for Web Services Adaptability Using Policies and Aspects, *International Journal of Cooperative Information Systems (IJCIS),* **15**(3), pp. 415-438, 2006.

[236] N. Pessemier, L. Seinturier, T. Coupaye, and L. Duchien, "A model for developing componentbased and aspect-oriented systems", *In Software Composition*, pages 259–274. Springer, 2006.

[237] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.B. Stefani, "The fractal component model and its support in java", *Software : Practice and Experience*, 36(11-12) :1257–1284, 2006.

[238] J.Y. Tigli, S. Lavirotte, G. Rey, V. Hourdin, N. Ferry, C. Vergoni, and M. Riveill, "Low Response Time Context-Awareness through Extensible Parameter Adaptation with ORCA", *Annals of Telecommunications*, 2012.

[239] D. Sacchetti, Y. Bromberg, N. Georgantas, V. Issarny, J. Parra, and R. Poortinga, "The Amigo Interoperable Middleware for the Networjked Home Environment", *in 6th International Middleware Conference, Workshops Proceedings,* Grenoble, France, 2005.

[240] N.Ferry, S.Lavirotte, G. Rey, and J.Y.Tigli, "Adaptation dynamique d'applications au context en informatique ambiante", *Rapport de recherché ISRN I3S/RR-2008-20-FR*, Octobre 2008.

[241] M. Janse, F. Ramparany, B. Kladis, L. Rozendaal, T. Broens, and H. Eertink, "Specification of the amigo abstract system architecture," *IST Amigo Project Deliverable D2.3.*

[242] J. Sousa and D. Garlan. "Aura: An Architectural Framework for User Mobility in Ubiquitous Computing Environments," *Software Architecture: System Design, Development and Maintenance*, 2002.

[243] J. Sousa and D. Garlan. "From Computers Everywhere to Tasks Anywhere: The Aura Approach," *School of Computer Science*, Carnegie Mellon University, http://www.cs.cmu.edu/aura.

[244] L. da Silva, P. Vink, and R. Poortinga-van Wijnen. "A Service-Oriented Middleware for Providing Context Awareness and Notification".

[245] S. Dehousse, S. Faulkner, C. Herssens, I. J. Jureta, and M. Saerens, "Learning optimal web service selections in dynamic environments when many quality-of-service criteria matter", *in Machine Learning*, Croatia: InTech, Feb. 2009, pp. 207–229.

[246] J. Gaber. "*New Paradigms for Ubiquitous and Pervasive Computing", Research Rep. No. RR-09-00.* Université de Technologies de Belfort-Montbéliard (UTBM), France. 2000.

[247] J. Gaber. "New Paradigms for Ubiquitous and Pervasive Applications", In *Proceedings of the 1st Workshop on Software Engineering Challenges for Ubiquitous Computing*, Lancaster, UK. 2006.

[248] T. Itao, T. Nakamura, M. Matsuo, T. Suda, and T. "Aoyama. Adaptive Creation of Network Applications in the Jack-in-the-Net Architecture". *IFIP Networking*, pp. 129-140. 2002.

[249] T. Itao, T. Nakamura, M. Matsuo, T. Suda, and T. Aoyama. "Service Emergence Based on Cooperative Interaction of Self-Organizing Entities". In *Proceedings of the IEEE Symposium on Applications and the Internet* (pp. 194-203), Nara, Japan. 2002.

[250] M. Bakhouya. "Self-adaptive Approach Based on Mobile Agent and Inspired by Human Immune System for Service Discovery in Large Scale Networks". *Doctoral thesis, Université de Technologies de Belfort-Montbeliard (UTBM)*. 2005.

[251] M. Bakhouya, and J. Gaber. "Adaptive Approaches for Ubiquitous Computing". In (Eds.), *Mobile Networks and Wireless Sensor Networks* (pp. 129-163, ISBN: 2-7462-1292-7). Hermes Science. 2006.

[252] S. Ben Mokhtar, N. Georgantas, and V. Issarny. "COCOA: COnversation-based service COmposition in pervAsive computing environments with QoS support" *in the Journal of Systems and Software* 80(12) 1941–1955. 2007.

[253] P-G. Raverdy, V. Issarny, R. Chibout, and A. de La Chapelle. "A multi-protocol approach to service discovery and access in pervasive environments", *In Proceedings of MOBIQUITOUS – The 3rd Annual International Conference on Mobile and Ubiquitous Systems: Networks and Services*.

[254] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut. "Quality of Service for workflows and Web service processes", *Journal of Web Semantic*, 2004.

[255] S. Ben Mokhtar, A. Kaul, N. Georgantas, and V. Issarny. "Efficient semantic service discovery in pervasive computing environments", *In: Proceedings of ACM/IFIP/USENIX 7th International Middleware Conference (Middleware'06).* 2006.

[256] A. Saffiotti, M. Broxvall, B. Seo, Y. Cho. "Steps toward an ecology of physically embedded intelligent systems", *in Proc of the 3rd Int Conf on Ubiquitous Robots and Ambient Intelligence,* Seoul, Korea, 2006.

[257] D. Guarino and A. Saffiotti. "On interfacing with an ubiquitous robotic system", In *Proc of the European Conf on AI*, pages 857–858, Riva del Garda, IT, 2006.

[258] S. Coradeschi and A. Saffiotti. "An introduction to the anchoring problem", *Robotics and Autonomous Systems*, 43(2-3):85–96, 2003.

[259] K. LeBlanc and A. Saffiotti. "Issues of perceptual anchoring in an ubiquitous robotic system", In *Proc of the ICRA-07 Workshop on Omniscient Space*, Rome, Italy, 2007.

[260] R. Lundh, L. Karlsson, and A. Saffiotti. "Plan-based configuration of an ecology of robots", In *Proc of the IEEE Int Conf on Robotics and Automation*, Rome, Italy, 2007.

[261] M. Gritti, M. Broxvall, and A. Saffiotti. "Reactive self-configuration of an ecology of robots", In *Proc of the ICRA-07 Workshop on Network Robot Systems*, Rome, Italy, 2007.

[262] A. Loutfi, M. Broxvall, S. Coradeschi, and A. Saffiotti. "An ecological approach to odour recognition in intelligent environments", In *Proc of the IEEE Int Conf on Robotics and Automation*, Orlando, FL, 2006.

[263] M. Broxvall, M. Gritti, A. Saffiotti, B.S. Seo, and Y.J. Cho. "PEIS ecology: Integrating robots into smart environments", In *Proc of the IEEE Int Conf on Robotics and Automation*, Orlando, FL, 2006.

[264] D. Arregui, C. Fernstrom, F. Pacull, and J. Gilbert. "STITCH: Middleware for ubiquitous applications.", In *Proc of the Smart Object Conf*, Grenoble, France, 2003.

[265] F. Siegemund. A context-aware communication platform for smart objects. In *Proc of the Int Conf on Pervasive Computing*, 2004.

[266] S. A. McIlraith, T. C. Son, and H. Zeng, "The Semantic Web Services", *IEEE Intelligent Systems*, Vol. 16, Issue 2, IEEE, pp. 46-53, 2001.

[267] D. Booth, H. Haas, F. McCabe, et al, "Web Services Architecture," http://www.w3c.org/TR/ws-arch/, *W3C Recommendation*, 2004.

[268] F. Manola, E. Miller, et al, "RDF Primer," http://www.w3c.org/TR/rdfprimer/, *W3C Recommendation*, 2004.

[269] M. Smith, et al, "OWL Web Ontology Language Guide," http://www.w3c.org/TR/owl-guide/, W3C Recommendation, 2004.

[270] K. Erol, D. Nau, and J. Hendler, "UMCP: A Sound and Complete Planning Procedure for HTN Planning," *AIPS-94*, Chicago, 1994.

[271] D. S. Nau, T. C. Au, et al, "SHOP2: An HTN Planning System," *J. of Artificial Intelligence Research, Vol.* 20, AIAF, 2003, pp. 379-404.

[272] T. Andrews, F. Curbera, H. Dholakia, et al, "BPEL for Web Services," http://www.ibm.com/developerworks/library/ws-bpel/, 2003.

[273] Evolution Robotics, "ERSP Scorpion," http://www.evolution.com/products/ersp/scorpion.masn.

[274] M. Paolucci, T. Kawamura, T.R. Payne, and K. Sycara. "Semantic matching of web services capabilities". *The semantic Web - ISWC,* 2342/2002, 2002.

[275] M. Caporuscio, P-G. Raverdy, and V. Issarny. "ubiSOAP: A Service Oriented Middleware for Ubiquitous Networking", *Journal of Transactions on Service Computing.* 2009.

[276] N. Georgantas, V. Issarny, S. Ben Mokhtar, Y-D. Bromberg, S. Bianco, G. Thomson, P-G. Raverdy, A. Urbieta, and R. Speicys Cardoso. "Middleware Architecture for Ambient Intelligence in the Networked Home". *In Nakashima, H., Augusto, J. C. & Aghajan, H. (Eds.), Handbook of Ambient Intelligence and Smart Environments.* Springer, 2009.

[277] O. Davidyuk, I. Selek, J.I. Duran, and J. Riekki. "Algorithms for Composing Pervasive Applications", *International Journal of Software Engineering and Its Applications*, 2 (2), 71-94. 2008.

[278] P. Wisner, D.N. Kalofonos. "A Framework for End-User Programming of Smart Homes Using Mobile Devices", *Proceedings of the 4th IEEE Consumer Communications and Networking Conference CCNC'07* (pp. 716-721), Washington DC: IEEE Computer Society. 2007.

[279] R. Rouvoy, et al. "Composing Components and Services using a Planning-based Adaptation Middleware", *In 7th Int. Symp. on Software Composition (SC),* LNCS 4954, Springer, 2008.

[280] G. Brataas, et al. "Scalability of Decision Models for Dynamic Product Lines". *In Int. Work. on Dynamic Software Product Line (DSPL).* 2007.

[281] S. A. Lundesgaard, et al. "Construction and Execution of Adaptable Applications Using an Aspect-Oriented and Model Driven Approach", *In 7th Int. Conf. on Distributed Applications and Interoperable Systems (DAIS),* LNCS 4531, Springer, 2007.

[282] K. Geihs, et al. "A comprehensive solution for application-level adaptation", *Software: Practice and Experience,* 2008

[283] M. U. Khan, R. Reichle, and K. Geihs. "Architectural Constraints in the Model-Driven Development of Self-Adaptive Applications", *IEEE Distributed Systems Online* 9(7), 2008.

[284] A. Dan, H. Ludwig, and G. Pacifici. "Web service differentiation with service level agreements". *IBM White Paper*. 2003.

[285] J. S. Rellermeyer and M. A. Kuppe. *jSLP*. http://jslp.sourceforge.net

[286] M. Demuru, F. Furfari, S. Lenzi. *DomoWare*. http://domoware.isti.cnr.it

[287] S. Kalasapur, M. Kumar, and B. Shirazi, "Personalized Service Composition for Ubiquitous Multimedia Delivery," *Proc. Sixth IEEE Int'l Symp. World of Wireless Mobile and Multimedia Networks* (WoWMoM '05), pp. 258-263, 2005.

[288] S. Kalasapur, M. Kumar, and B. Shirazi, "Seamless Service Composition (SESCO) in Pervasive Environments," *Proc. First ACM Int'l Workshop Multimedia Service Composition,* pp. 11-20, 2005.

[289] M. Kumar, B.A. Shirazi, S.K. Das, M. Singhal, B. Sung, and D. Levine, "Pervasive Information Communities Organization PICO: A Middleware Framework for Pervasive Computing," *IEEE Pervasive Computing,* vol. 2, pp. 72-79, 2003.

[290] G. A. Mann, "BEELINE—a situated, bounded conceptual knowledge system", *Int. J. Syst. Res. Inf. Sci.*, vol. 7, pp. 37–53, 1995.

[291] K. Fujii and T. Suda. "Dynamic service composition using semantic information", *ICSOC 2004*: page 39-48, 2004.

[292] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, "Tropos: An agent-oriented software development methodology," *Autonomous Agents and Multi-Agent Systems*, vol. 8, no. 3, pp. 203–236, May 2004.

[293] M. Jang and J. Sohn, "Bossam: An extended rule engine for OWL inferencing," in *Proc. RuleML*, vol. 3323, LNCS, pp. 128–138, Nov. 2004.

[294] C. Forgy, "RETE: A fast algorithm for the many pattern/many object pattern match problem," *Artif. Intell.*, no. 19, pp. 17–37, 1982.

[295] A. Michlmayr, F. Rosenberg, C. Platzer, M. Treiber, and S. Dustdar, "Towards Recovering the Broken SOA Triangle – A Software Engineering Perspective," *in Proceedings of the 2nd International Workshop on Service Oriented Software Engineering*, Dubrovnik, Croatia, pp. 22–28, 2007.

[296] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar, "Advanced Event Processing and Notifications in Service Runtime Environments*," in Proceedings of the 2nd International Conference on Distributed Event-Based Systems (DEBS'08)*, Rome, Italy. ACM, pp. 115–125, 2008.

[297] F. Rosenberg, P. Leitner, A. Michlmayr, and S. Dustdar, "Integrated Metadata Support for Web Service Runtimes," *in Proceedings of the Middleware for Web Services Workshop (MWS'08), co-located with the 12th IEEE International Distributed Object Computing Conference (EDOC'08),* Munich, Germany. IEEE Computer Society, Sept. 2008.

[298] F. Rosenberg, C. Platzer, and S. Dustdar, "Bootstrapping Performance and Dependability Attributes of Web Services," *in Proceedings of the IEEE International Conference on Web Services (ICWS'06*), Chicago, USA, Sept. 2006.

[299] R. Eshuis, P. W. P. J. Grefen, and S. Till, "Structured service composition," *in Proceedings of the 4th International Conference on Business Process Management,* Vienna, Austria, pp. 97–112, 2006.

[300] M. Blay-Fornarino, A. Charfi, D. Emsellem, A-M. Pinna-Dery, and M. Riveill. "Software interactions", *J Object Technol* 3(10):161–180, 2004.

[301] D. Cheung-Foo-Wo, J-Y. Tigli, S. Lavirotte, and M. Riveill. "Wcomp: a multi-design approach for prototyping applications using heterogeneous resources", *In: 17th IEEE int. workshop on rapid syst. prototyping*, Crete, pp 119–125, 2006

[302] D. Spiegelhalter, A. Dawid, S. Lauritzen, and R. Cowell. "Bayesian analysis in expert system". 1993.

[303] K. Kaemarungsi, "Design of Indoor positioning system based on location fingerprint techinique". University of Pittsburgh, 2005.